

Sandro Rogério Silva de Carvalho

*Modelagem do Processador PowerPC405
para uma Plataforma de SoCs*

Florianópolis – SC

Fevereiro / 2007

Sandro Rogério Silva de Carvalho

*Modelagem do Processador PowerPC405
para uma Plataforma de SoCs*

Monografia apresentada ao curso de Bacharelado em Ciências da Computação da Universidade Federal de Santa Catarina como requisito parcial para obtenção do grau de Bacharel em Ciências da Computação.

Orientador:

Prof. Dr. Luiz Cláudio Villar dos Santos

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO

Florianópolis – SC

Fevereiro / 2007

Monografia de graduação sob o título “*Modelagem do processador PPC405 para uma plataforma de SoCs*”, defendida por Sandro Rogério Silva de Carvalho e aprovada em 7 de fevereiro de 2007, em Florianópolis , Santa Catarina, pela banca examinadora constituída pelos professores:

Prof. Dr. Luiz Cláudio Villar dos Santos
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Luís Fernando Friedrich
Universidade Federal de Santa Catarina
Membro da Banca

Prof. Dr. Olinto José Varela Furtado
Universidade Federal de Santa Catarina
Membro da Banca

*Dedico esta monografia
à minha amada esposa Juliana.*

“A diferença entre o possível e o impossível está na vontade humana.”

Louis Pasteur

Resumo

A crescente densidade de transistores que podem ser feitos em um chip de silício, possibilitou o surgimento de sistemas inteiros em um único chip, chamados *System-on-Chip* (SoCs), que agregam tanto o hardware quanto o software necessários para garantir a funcionalidade do sistema.

Fazer um chip específico demanda um alto investimento de capital e desenvolver o software para um sistema único é inviável, pois dificulta um fator primordial de aumento de produtividade, o reuso de programas já confeccionados e testados. Assim nasceu o conceito de plataforma, que visa o reuso de hardware, através de componentes denominados IP's (Intellectual Property), e de software para desenvolver novos sistemas. Os IP's são inseridos e retirados do sistema conforme a necessidade do desenvolvedor e possibilita a exploração de várias alternativas de implementação.

Este trabalho apresenta a descrição do comportamento do processador PowerPC 405, com forte presença no mercado de sistemas embutidos, em uma Linguagem de Descrição de Arquiteturas (ArchC) para ser integrada como uma alternativa de unidade de processamento na plataforma ARP (ArchC Reference Plattform).

Lista de Figuras

2.1	Registradores de Usuário - PowerPC405	p. 14
2.2	Formato de Instrução I.	p. 16
2.3	Formato de Instrução B.	p. 17
2.4	Formato de Instrução SC.	p. 17
2.5	Formato de Instrução D.	p. 18
2.6	Formato de Instrução X.	p. 19
2.7	Formato de Instrução XL.	p. 20
2.8	Formato de Instrução XFX.	p. 20
2.9	Formato de Instrução XO.	p. 21
2.10	Formato de Instrução M.	p. 21
2.11	Funcionamento de uma instrução de Multiplicação-Soma	p. 21
3.1	Descrição dos parâmetros do processador no modelo funcional.	p. 25
3.2	Informações de codificação das instruções.	p. 26
3.3	Descrição do comportamento das instruções add e macchw no modelo funcional.	p. 27
3.4	Descrição dos parâmetros do pipeline.	p. 28
3.5	Descrição do comportamento da instrução add no modelo com precisão de ciclos.	p. 29
4.1	Metodologia de Validação	p. 34
4.2	Número de instruções de cada benchmark do MiBench nas arquiteturas consideradas.	p. 36

4.3	Porcentagem de instruções executadas por aplicação do MiBench no modelo funcional do MIPS e do NIOS2 em relação ao modelo funcional do PowerPC405.	p. 37
4.4	Porcentagem do tempo de simulação para gerar os resultados no modelo funcional do MIPS e do NIOS2 em relação aos do modelo funcional do PowerPC405.	p. 38
4.5	Média da porcentagem dos dados coletados do modelo funcional do MIPS e do NIOS2 em relação aos do modelo funcional do PowerPC405. . . .	p. 39
4.6	Porcentagem de instruções executadas por aplicação do MiBench nos modelos com precisão de ciclos do MIPS e do NIOS2 em relação ao modelo com precisão de ciclos do PowerPC405.	p. 40
4.7	Porcentagem do tempo de simulação para gerar os resultados nos modelos com precisão de ciclos do MIPS e do NIOS2 em relação aos do modelo com precisão de ciclos do PowerPC405.	p. 40
4.8	Média da porcentagem dos dados coletados dos modelos com precisão de ciclos do MIPS e do NIOS2 em relação aos do modelo com precisão de ciclos do PowerPC405.	p. 41
4.9	Porcentagem de instruções executadas/buscadas por aplicação do MiBench no modelo com precisão de ciclos PowerPC405 em relação ao modelo funcional deste mesmo processador.	p. 41
4.10	Porcentagem do tempo de simulação para gerar os resultados no modelo funcional do MIPS e do NIOS2 em relação aos do modelo funcional do PowerPC405.	p. 42

Lista de Tabelas

2.1	Categorias do Conjunto de Instruções.	p. 16
2.2	Instruções de multiplicação-soma e de multiplicação de meia-palavra.	p. 22
4.1	Aplicativos do MiBench	p. 31
4.2	Conjunto de programas Mibench - PowerPC405 Funcional	p. 44
4.3	Conjunto de programas Mibench - MIPS Funcional	p. 45
4.4	Conjunto de programas Mibench - NIOS2 Funcional	p. 46
4.5	Conjunto de programas Mibench - PowerPC405 com Precisão de Ciclos	p. 47
4.6	Conjunto de programas Mibench - MIPS com Precisão de Ciclos	p. 47
4.7	Conjunto de programas Mibench - NIOS2 com Precisão de Ciclos	p. 48

Sumário

1	Introdução	p. 11
2	O Processador PowerPC405	p. 13
2.1	Visão Geral	p. 13
2.2	Registradores	p. 14
2.3	Conjunto de Instruções	p. 15
2.4	Modos de Endereçamento	p. 16
2.4.1	Modos de Endereçamento de Carga e Armazenamento	p. 16
2.4.2	Modos de Endereçamento de Desvio	p. 17
2.5	Ponto-Flutuante	p. 19
2.6	A Unidade de Multiplicação-Soma	p. 21
3	Estrutura dos Modelos	p. 23
3.1	A Linguagem ArchC	p. 23
3.2	Modelo Puramente Funcional	p. 24
3.3	Modelo com Precisão de Ciclos	p. 28
3.4	Dificuldades Encontradas	p. 29
4	Resultados Experimentais	p. 31
4.1	Configuração Experimental	p. 31
4.2	Validação	p. 33
4.2.1	Modelo Funcional	p. 33
4.2.2	Modelo com Precisão de Ciclos	p. 34

4.3	Comparação de Desempenho	p. 34
4.3.1	PPC405 funcional X MIPS funcional X NIOS2 funcional	p. 35
4.3.2	PPC405 ca X MIPS ca X NIOS2 ca	p. 36
4.3.3	PPC405 ca X PPC405 funcional	p. 38
5	Conclusão	p. 49
5.1	Produtos Gerados	p. 49
5.2	Trabalhos Futuros	p. 50
	Referências	p. 51
	Anexo A - Código Fonte	p. 53
5.1	Funcional	p. 53
5.1.1	ppc405.ac	p. 53
5.1.2	ppc405_isa.ac	p. 54
5.1.3	ppc405_syscall.cpp	p. 77
5.1.4	ppc405_gdb_funcs.cpp	p. 80
5.1.5	ppc405-isa.cpp	p. 84
5.2	Precisão de Ciclos	p. 166
5.2.1	ppc405-ca.ac	p. 166
5.2.2	ppc405-ca_isa.ac	p. 168
5.2.3	ppc405-ca_syscall.cpp	p. 190
5.2.4	ppc405-ca-isa.cpp	p. 193
	Anexo B - Artigo	p. 194

1 *Introdução*

O aumento do número de transistores que podem ser fabricados em uma dada área de silício vem aumentando conforme a previsão empírica que Gordon Moore propôs em 1965, dizendo ser possível dobrar esse número a cada vinte e quatro meses (Moore 1965). Isto aliado à excessiva demanda por aplicações cada vez mais complexas da indústria de sistemas embarcados impulsionou o desenvolvimento de sistemas inteiros em um único chip, os *System-on-Chip* (SoCs). Esses sistemas envolvem a combinação de *hardware* (processadores, memórias e periféricos) e *software* (*Hardware Dependable Software* e aplicação).

Fazer hardware e software exclusivo para um dado sistema é inviável devido ao alto custo envolvido na fabricação de máscaras para a construção do chip e à potencial perda ao não se reutilizar o software. Esse problema motivou o surgimento do projeto baseado em plataformas (Sangiovanni-Vincentelli e Martin 2001), que seria a utilização de uma arquitetura de referência (biblioteca de componentes reutilizáveis), envolvendo processadores, memórias e periféricos, e também os programas já utilizados, adaptando-os às necessidades do sistema a ser projetado. Isso diminui os custos envolvidos e acelera o tempo que o produto chega ao mercado.

Com a utilização de plataformas surgiram as linguagens para descrevê-las, chamadas de Linguagens de Descrição de Sistemas (SDL - *System Description Language*) (Keutzer et al. 2000), como é o caso de SystemC (SystemC), que pode ser utilizada em diversos níveis de descrição, desde o Nível de Sistema até o Nível de Transferência de Registradores (RTL - *Register Transfer Language*), que é bem próximo do comportamento do hardware e possibilita a síntese automática destes. Com a utilização de uma linguagem para descrever o sistema espera-se facilidade de exploração de soluções alternativas, com avaliação de corretude, desempenho e consumo de energia.

Aqui descrevemos um processador a ser utilizado na exploração de alternativas que possui a terceira mais vendida arquitetura de instruções, a do PowerPC. O PowerPC405 é bastante utilizado em sistemas embarcados, geralmente com a função de controle das

operações do sistema, mas também é muito utilizado como unidade principal de processamento.

Este modelo é feito em ArchC (ArchC 2005), uma Linguagem de Descrição de Arquiteturas e será utilizado como alternativa na plataforma de referência ArchC, denominada ARP (ArchC Reference Plattform).

No capítulo dois é dada uma visão geral do processador modelado e no três mostramos como foi feita a modelagem. O capítulo quatro mostra resultados experimentais e comparações com outros modelos de processadores populares e, por fim, o capítulo cinco conclui o trabalho e vislumbra possíveis pesquisas futuras.

2 *O Processador PowerPC405*

Este capítulo apresenta algumas características do processador PowerPC405. Os dados foram retirados de (Xilinx 2003) e (IBM 2005).

2.1 Visão Geral

O PowerPC405 é um processador RISC *load-store* 32 bits largamente utilizado em sistemas embarcados. Ele é uma implementação da arquitetura PowerPC para ambiente-embutido. Garante alta performance e baixo consumo de energia às aplicações embutidas de ponto-fixa. É compatível com PowerPC UISA (*User Instructions Set Architecture*) e pode ser utilizado tanto como uma máquina *big-endian* quanto *little-endian*.

Faz uso de uma Unidade de Multiplicação-Soma (*MAC - Multiply-Accumulate Unit*) e não possui unidade de ponto-flutuante

A arquitetura PowerPC foi criada através da aliança AIM entre as empresas: Apple, IBM e Motorola. Foi concebida para substituir os obsoletos chips 68000 da Motorola que até então residiam nos computadores fabricados pela Apple. A IBM e a Motorola voltaram-se para o mercado de processadores embarcados e a Apple ficou esperando o seu chip para Desktop de alta performance, capaz de competir com os famosos chips da Intel.

Nem a IBM, nem a Motorola, atual Freescale, satisfizeram o desejo de Steve Jobs da Apple, principalmente pela alta dissipação de calor nos PowerPC's de alto desempenho. A Apple resolveu então, se render à gigante Intel, e começou a usar seus processadores. A IBM e várias outras empresas criaram o Power.org a fim de aperfeiçoar a arquitetura Power com a criação de uma plataforma aberta de desenvolvimento de hardware denominada PAPR (*Power Architecture Platform Reference*).

O PowerPC405 é um dos muitos processadores fabricados ou licenciados para fabricação pela IBM. É também utilizado como unidade de processamento central (*core*) de sistemas embarcados a serem inseridos em chips de FPGA da Xilinx.

2.2 Registradores

A figura 2.1 mostra os registradores de usuário suportados pelo PowerPC405, todos eles estão disponíveis ao software em execução tanto no modo usuário quanto no modo supervisor. No PowerPC405, todos os registradores de usuário têm 32-bits, exceto os de base de tempo. Registradores de ponto-flutuante não são suportados pelo PowerPC405.

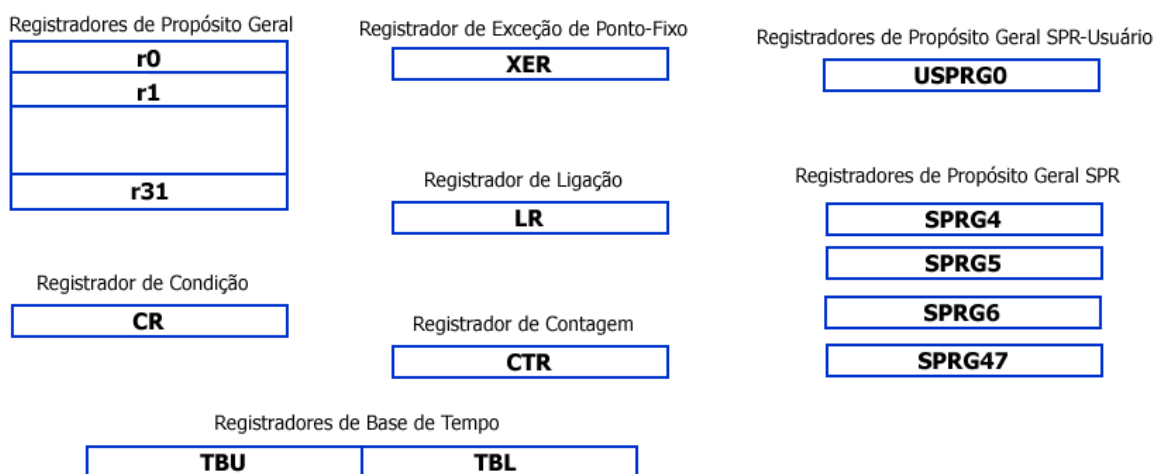


Figura 2.1: Registradores de Usuário - PowerPC405

Registradores de Propósito Especial (SPRs) SPRs controlam a operação de depuração, temporizadores, interrupções, atributos do controle de armazenamento, e outros recursos do processador. Podem ser acessados explicitamente usando instruções de movimento (mfspr e mtspr).

Registradores de Propósito Geral (GPRs) O PowerPC405 tem trinta e dois registradores de propósito geral de 32-bits, numerados de r0 a r31. Onde são gravados dados da memória em uma instrução de carga e seu conteúdo é gravado na memória em uma instrução de armazenamento. Muitas instruções inteiras usam os GPRs como operandos de fonte e destino da operação.

Registrador de Condição (CR) Reflete o resultado de certas instruções provendo mecanismo para teste e desvio condicional. Os bits no CR são agrupados em oito campos de 4bits. Os do campo indicam se o resultado foi negativo, positivo, igual a zero e se houve overflow.

Registrador de Exceção de Ponto-Fixo (XER) Reflete o resultado de operações aritméticas que resultaram em overflow ou carry. Também é usado para indicar o número de bytes a ser transferido em instruções de carga-armazenamento indexadas.

Registrador de Ligação (LR) Usado em instruções de desvio, geralmente para ligar subrotinas. Dois tipos de instruções usam o LR:

- Desvio condicional para registrador de ligação(bclrx) lêem o endereço-alvo do LR
- Desvio com opção de atualização do LR habilitada salva o endereço da instrução atual mais quatro no LR

Registradores de Propósito Geral SPR-Usuário(USPRG0) Pode ser usado pela aplicação com qualquer propósito. O valor gravado neste registrador não afeta a operação do processador.

Registradores de Propósito Geral SPR (SPRG) Podem ser usados pelo sistema com qualquer propósito. A aplicação em modo usuário pode ler, porém não pode escrever neles. O valor gravado nestes registradores não afeta a operação do processador.

Registradores de Base de Tempo (TBU-TBL) É um contador de incremento de 64-bits implementado como dois registradores de 32-bits. O superior(TBU) contém os bits 0:31, e o inferior (TBL), os bits 32:63.

2.3 Conjunto de Instruções

Todas instruções têm quatro bytes e são alinhadas. Na tabela 2.1 vemos as categorias do conjunto de instruções deste processador.

Bits 0:5 sempre contém o opcode primário, que é usado para determinar o formato da instrução. O formato da instrução define campos para identificar os operandos. Alguns formatos de instruções definem um campo de opcode estendido para especificar instruções adicionais.

Existem nove formatos de instruções que estão ilustrados nas figuras 2.2 até 2.10, retiradas de (IBM 2005). Campos com barras (/, // ou ///) são reservados.

Tabela 2.1: Categorias do Conjunto de Instruções.

Categoria	Instruções
Referência à Memória	load, store
Aritmética e Lógica	add, subtract, negate, multiply, divide, and, andc, or, orc, xor, nand, nor, xnor, sign extension, count leading zeros, multiply accumulate
Comparação	compare, compare logical, compare immediate
Desvio	branch, branch conditional, branch to LR, branch to CTR
Lógica CR	crand, crandc, cror, crorc, crnand, crnor, crxor, crxnor, move CR field
Rotate/Shift	rotate and insert, rotate and mask, shift left, shift right
Controle de Cache	invalidate, touch, zero, flush, store, read
Controle de Interrupção	write to external interrupt enable bit, move to/from MSR, return from interrupt, return from critical interrupt
Gerenciamento do Processador	system call, synchronize, trap, move to/from DCRs, move to/from SPRs, move to/from CR

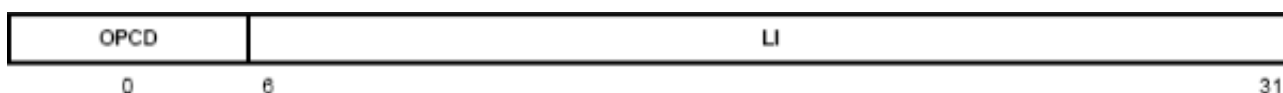


Figura 2.2: Formato de Instrução I.

2.4 Modos de Endereçamento

Nesta seção examinaremos os modos de endereçamento do PowerPC405 que são divididos em dois grupos:

- Modos de Endereçamento de Carga e Armazenamento;
- Modos de Endereçamento de Desvio.

2.4.1 Modos de Endereçamento de Carga e Armazenamento

O PowerPC405 suporta os seguintes modos de endereçamento que possibilitam eficiente carga e armazenamento de dados na memória:

- Endereçamento base mais deslocamento;
- Endereçamento indexado;
- Endereçamento base mais deslocamento e endereçamento indexado, com atualização.



Figura 2.3: Formato de Instrução B.



Figura 2.4: Formato de Instrução SC.

No modo de endereçamento base mais deslocamento, o endereço de desvio é formado somando-se o deslocamento ao endereço base contido em um Registrador de Propósito Geral (ou a uma base igual a zero se o registrador for igual a zero). O deslocamento é um campo imediato na instrução.

No modo de endereçamento indexado, o endereço de desvio é formado somando-se o conteúdo de dois registradores de propósito geral, um é o índice e o outro é a base (que será zero se o registrador for zero).

Os modos de endereçamento base mais deslocamento e o indexado, também podem prover uma atualização da base. No modo “com atualização”, o endereço de desvio calculado é salvo no registrador base da instrução atual, que pode ser utilizado como base na próxima instrução. O modo “com atualização” libera o processador de carregar um registrador com um endereço para cada dado a ser carregado, levando-se em conta a proximidade dos dados na memória.

2.4.2 Modos de Endereçamento de Desvio

Na execução de instruções sequenciais, o próximo endereço é calculado adicionando-se quatro bytes ao endereço da instrução atual. No caso de instruções de desvio, entretanto, o endereço da próxima instrução é determinado usando um dos quatro modos de endereçamento de desvio:

- Desvio Relativo (condicional e incondicional)
- Desvio Absoluto (condicional e incondicional)
- Desvio para o Registrador de Ligação - Link Register (somente condicional)
- Desvio para o Registrador de Contagem - Count Register (somente condicional)

OPCD	RT			RA	D
OPCD	RS			RA	SI
OPCD	RS			RA	D
OPCD	RS			RA	UI
OPCD	BF	/	L	RA	SI
OPCD	BF	/	L	RA	UI
OPCD	TO			RA	SI
0	6	11	16		31

Figura 2.5: Formato de Instrução D.

No desvio incondicional relativo, o endereço da próxima instrução é gerado através da adição de dois bits zero ao operando deslocamento-imediato (LI) e extendendo-se o sinal do resultado, que é então, adicionado ao endereço da instrução atual.

No desvio condicional relativo, se a condição do desvio é atendida, o endereço da próxima instrução é gerado através da adição de dois bits zero ao operando deslocamento-imediato (BD) e extendendo-se o sinal do resultado, que é então, adicionado ao endereço da instrução atual.

No desvio incondicional absoluto, o endereço da próxima instrução é gerado através da adição de dois bits zero ao operando deslocamento-imediato (LI) e extendendo-se o sinal do resultado.

No desvio condicional absoluto, se a condição do desvio é atendida, o endereço da próxima instrução é gerado através da adição de dois bits zero ao operando deslocamento-imediato (BD) e extendendo-se o sinal do resultado.

No desvio condicional para o registrador de ligação, se a condição do desvio é atendida, o endereço da próxima instrução é gerado através da leitura do registrador de ligação (LR) e colocando-se zero nos dois bits de mais baixa ordem.

No desvio condicional para o registrador de contagem, se a condição do desvio é atendida, o endereço da próxima instrução é gerado através da leitura do registrador de contagem (CTR) e colocando-se zero nos dois bits de mais baixa ordem.

Em todos os modos de endereçamento de desvio, a atualização do registrador de ligação é habilitada se o campo LK da instrução for igual a 1. Esta opção faz com que o endereço da próxima instrução que segue a atual, ou seja, a atual mais 4, seja salvo no registrador de ligação (LR).

OPCD	RT	RA	RB	XO	Rc		
OPCD	RT	RA	RB	XO	/		
OPCD	RT	RA	NB	XO	/		
OPCD	RT	RA	WS	XO	/		
OPCD	RT	///	RB	XO	/		
OPCD	RT	///	///	XO	/		
OPCD	RS	RA	RB	XO	Rc		
OPCD	RS	RA	RB	XO	1		
OPCD	RS	RA	RB	XO	/		
OPCD	RS	RA	NB	XO	/		
OPCD	RS	RA	WS	XO	/		
OPCD	RS	RA	SH	XO	Rc		
OPCD	RS	RA	///	XO	Rc		
OPCD	RS	///	RB	XO	/		
OPCD	RS	///	///	XO	/		
OPCD	BF	/ L	RA	RB	XO	/	
OPCD	BF	//	BFA	//	///	XO	Rc
OPCD	BF	//	///	///	XO	/	
OPCD	BF	//	///	U	XO	Rc	
OPCD	BF	//	///	///	XO	/	
OPCD	TO	RA	RB	XO	/		
OPCD	BT	///	///	XO	Rc		
OPCD	///	RA	RB	XO	/		
OPCD	///	///	///	XO	/		
OPCD	///	///	E	//	XO	/	
0	6	11	16	21	31		

Figura 2.6: Formato de Instrução X.

2.5 Ponto-Flutuante

O PowerPC405 é um processador de números inteiros. Ou seja, não suporta a execução de instruções de ponto-flutuante em hardware, porém, se necessitamos de utilizar a aritmética de ponto-flutuante, podemos fazer uso de sua emulação por software. Há duas formas de se fazer isto:

- Chamar funções de uma biblioteca, ou
- Utilizar interrupção.

OPCD	BT		BA		BB	XO	/
OPCD	BC		BI		///	XO	LK
OPCD	BF	//	BFA	//	///	XO	/
OPCD	///		///		///	XO	/
0	6	11	16	21	31		

Figura 2.7: Formato de Instrução XL.

OPCD	RT	SPRF			XO	/
OPCD	RT	DCRF			XO	/
OPCD	RT	/	FXM	/	XO	/
OPCD	RS	SPRF			XO	/
OPCD	RS	DCRF			XO	/
0	6	11	16	21	31	

Figura 2.8: Formato de Instrução XFX.

É mais aconselhável a utilização de uma interface de chamada de funções de uma biblioteca. Cada função pode emular a operação de uma instrução de ponto-flutuante. Este método requer a recompilação da aplicação para que as operações de ponto-flutuante se tornem chamadas à interface e para que se faça a ligação com a biblioteca.

Uma outra alternativa seria utilizar interrupções de programa. Quando se tenta executar instruções de ponto-flutuante no PowerPC405, ela é encarada como uma instrução ilegal e é gerada uma interrupção de programa. O tratador de interrupção deverá decodificar a instrução ilegal e chamar a função apropriada de uma biblioteca que emula a execução da instrução de ponto-flutuante utilizando instruções de inteiros. Este método é menos aconselhável devido à sobrecarga de processamento relativa à execução do tratador de interrupções. Entretanto, ele suporta a execução de programas que contém instruções de ponto-flutuante do PowerPC, não necessitando ser recompilado.

Este trabalho utiliza o método de chamadas de função de uma biblioteca. Isto é feito passando-se um parâmetro para o compilador quando da compilação. Esta biblioteca faz parte das bibliotecas do compilador redirecionável utilizado. Mesmo se quizessemos utilizar a segunda abordagem não seria possível, pois o ArchC não possui suporte à interrupção na versão estável atual. No entanto já há estudos neste sentido.

OPCD	RT	RA	RB	OE	XO	Rc
OPCD	RT	RA	RB	OE	XO	Rc
OPCD	RT	RA	///	/	XO	Rc
0	6	11	16	21 22		31

Figura 2.9: Formato de Instrução XO.

OPCD	RS	RA	RB	MB	ME	Rc
OPCD	RS	RA	SH	MB	ME	Rc
0	6	11	16	21	26	31

Figura 2.10: Formato de Instrução M.

2.6 A Unidade de Multiplicação-Soma

Sempre que se trabalha com matrizes e vetores, duas operações vêm à tona, multiplicação e soma. (Hennessy e Patterson 2000)

A operação de multiplicação-soma basicamente lê três operandos, multiplica dois deles, soma o terceiro ao produto e escreve a soma no operando resultante. Conforme podemos ver na figura 2.11.

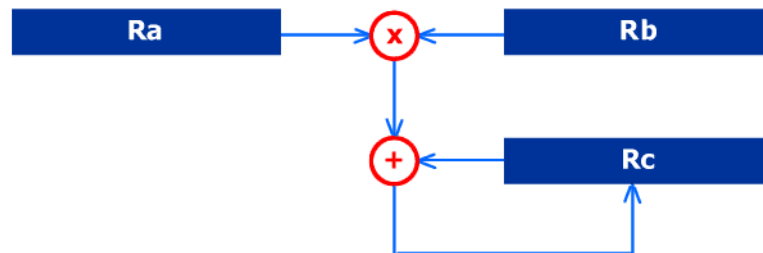


Figura 2.11: Funcionamento de uma instrução de Multiplicação-Soma

A instrução de multiplicação-soma realiza as duas operações e *depois* arredonda o resultado, ao contrário do que ocorre quando se executa uma instrução de multiplicação e uma de soma: neste caso o arredondamento ocorre após cada operação. Tais instruções também trabalham com bits extras para resultados intermediários, melhorando a precisão. Além de potencialmente mais rápidas, a instrução de multiplicação-soma também pode ser útil para o cálculo da divisão, da raiz quadrada e nas bibliotecas que trabalham com precisão de 64 bits. A possibilidade de uma divisão mais precisa foi a principal motivação para não se efetuar o arredondamento entre as duas operações.

Tabela 2.2: Instruções de multiplicação-soma e de multiplicação de meia-palavra.

Multiplicação-soma	Multiplicação-soma Negativa	Multiplicação de Meia Palavra
macchw[o][.]	nmacchw[o][.]	mulchw[.]
macchws[o][.]	nmacchws[o][.]	mulchwu[.]
macchwsu[o][.]	nmachhw[o][.]	mulhhw[.]
macchwu[o][.]	nmachhws[o][.]	mulhhwu[.]
machhw[o][.]	nmaclhw[o][.]	mullhw[.]
machhws[o][.]	nmaclhws[o][.]	mullhwu[.]
machhwsu[o][.]		
machhwu[o][.]		
maclhw[o][.]		
maclhws[o][.]		
maclhwsu[o][.]		
maclhwu[o][.]		

O PowerPC405 suporta um conjunto de instruções de multiplicação-soma de inteiros que provê funcionalidades úteis a certas aplicações que fazem uso intensivo do processador, tal como aquelas que implementam algoritmos de Processadores de Sinais Digitais (*digital signal processor - DSP*). Estas instruções são compatíveis com os requisitos exigidos para uma Unidade de Processamento Auxiliar (*auxiliary-processor unit - APU*) definida pela Arquitetura PowerPC para Ambiente-Embutido. Porém, são consideradas dependentes da implementação e não fazem parte da Arquitetura PowerPC Padrão e nem da Arquitetura PowerPC para Ambiente-Embutido. Ou seja, programas que utilizam estas instruções não executam em todas as implementações do PowerPC.

Conforme podemos conferir na tabela 2.2. O conjunto de instruções de multiplicação-soma incluem as instruções de multiplicação-soma, instruções de multiplicação-soma negativa e instruções de multiplicação de meia-palavra.

Na tabela 2.2, a sintaxe [o] indica que a instrução tem uma forma "o" que atualiza o XER[SO,OV] sinalizando *overflow*. E a sintaxe [.] indica que a instrução tem uma forma "record" que atualiza o CR[CR0] armazenando se o resultado foi maior, menor ou igual a zero.

3 *Estrutura dos Modelos*

Este capítulo descreve brevemente a linguagem de descrição de arquiteturas utilizada e a estrutura dos modelos construídos.

3.1 A Linguagem ArchC

Linguagem de Descrição de Arquitetura (*Architecture Description Language - ADL*) é a classificação dada a uma família de linguagens que surgiram como resposta à necessidade dos desenvolvedores de sistemas embarcados de estimar tempo de processamento, tamanho de código e consumo de energia em diversas arquiteturas diferentes para serem utilizadas como ferramenta de comparação, visto que essas variáveis são de extrema importância na construção de seus sistemas.

A utilização de uma ADL torna mais branda a obtenção de um modelo, um programa executável, que simule uma arquitetura, bem como a construção automática de toda a gama de software dependente de hardware necessária para sua devida utilização: montadores, ligadores, depuradores e compiladores. Podemos citar como exemplos de ADL's: ArchC, nML, ISDL, EXPRESSION e LISA. (Rigo et al. 2004)

Embora o termo “Linguagem de Descrição de Arquitetura” também seja usado para definir software ADL's que têm a função de ajudar na construção de arquiteturas de software, aqui nos referimos às hardware ADL's, utilizadas na construção de arquiteturas de sistemas computacionais.

A linguagem utilizada na descrição dos modelos deste trabalho foi a ArchC (ArchC 2005), desenvolvida no Laboratório de Sistemas Computacionais (LSC) do Instituto de Computação (IC) da Universidade de Campinas (UNICAMP), apresenta como principal vantagem a geração de simuladores em SystemC (SystemC), que conforme (Bhasker 2002) é uma biblioteca de classes de c++ que a torna capaz de modelar hardware em vários níveis de abstração através da adição de conceitos importantes como: concorrência, even-

tos temporizados e tipos de dados específicos. Segundo relatado em (Rigo et al. 2004), ArchC visa, principalmente, facilitar e acelerar a descrição de processadores e tem poder de expressão suficiente para modelar diversas classes de arquiteturas (RISC, CISC, DSPs, etc).

É possível gerar simuladores e montadores na atual versão estável de ArchC (1.6.0), porém na versão em desenvolvimento (2.0) já podemos gerar desmontadores, depuradores e ligadores a partir dos modelos de processadores.

A descrição de processadores em ArchC se divide em duas partes, provendo informações estruturais e comportamentais. A descrição da arquitetura do conjunto de instruções (AC_ISA) é onde o desenvolvedor provê detalhes relacionados ao formato, tamanho e nome das instruções, as informações necessárias para decodificá-las e o comportamento de cada instrução. Já a descrição dos recursos da arquitetura (AC_ARCH), informa ao ArchC sobre dispositivos de armazenamento, estrutura do pipeline, hierarquia de memória, etc. Baseada nestas duas descrições, ArchC irá gerar um simulador comportamental escrito em SystemC para a arquitetura, que pode ser puramente funcional ou com precisão de ciclos, dependendo do nível de abstração usado para descrever o comportamento das instruções. (Rigo et al. 2004)

3.2 Modelo Puramente Funcional

A descrição do modelo funcional do PowerPC405 foi realizada baseado-se no modelo que está descrito em (Casarotto e Filho 2004), extendendo-se o modelo funcional do PowerPC disponível em (ArchC 2005).

É composto de cinco arquivos: `ppc405.ac`, `ppc405_isa.ac`, `ppc405-isa.cpp`, `ppc405_syscall.cpp`, `ppc405_gdb_funcs.cpp`.

Partimos da descrição estrutural da arquitetura no arquivo `ppc405.ac`, conforme mostra a figura 3.2. Podemos notar a definição do tamanho da palavra do processador (`ac_wordsize`), da quantidade de memória disponível e nome que a referencia (`ac_mem`), quantidade de registradores de propósito geral e o nome do banco (`ac_regbank`), a declaração de outros registradores (`ac_reg`) e, por fim, a indicação do arquivo que contém a declaração das instruções (`ac_isa`) e a orientação dos bytes nas palavras (`ac_endian`).

No segundo arquivo, `ppc405_isa.ac`, cujo fragmento está na figura 3.2, descrevemos como as instruções são codificadas. Isto é feito da seguinte forma: declaramos os formatos

```
AC_ARCH(ppc405) {  
  
    ac_wordsize 32;  
  
    ac_mem MEM:8M;  
  
    ac_regbank GPR:32;  
  
    ac_reg SPRG4;  
    ac_reg SPRG5;  
    ac_reg SPRG6;  
    ac_reg SPRG7;  
    ac_reg USPRG0;  
  
    ac_reg XER;  
  
    ac_reg MSR;  
  
    ac_reg EVPR;  
    ac_reg SRR0;  
    ac_reg SRR1;  
  
    ac_reg CR;  
    ac_reg LR;  
    ac_reg CTR;  
  
    ARCH_CTOR(ppc405) {  
        ac_isa("ppc405_isa.ac");  
        set_endian("big");  
    };  
};
```

Figura 3.1: Descrição dos parâmetros do processador no modelo funcional.

de instruções (`ac.format`), detalhando seus campos, e declaramos as instruções indicando seu formato (`ac.instr`).

Dentro do construtor indicamos as formas alternativas de escrever uma instrução que influenciam no valor de seus campos, pois aqueles que não estiverem explícito no assembly (`set_asm`), são indicados como um campo constante da instrução (`ac.decoder`).

No exemplo da figura, a instrução `add` tem o formato `XO1` e possui quatro formas alternativas de ser escrita:

- `add` - Somente soma;
- `add.` - Soma com atualização do registrador de condição;

```

AC_ISA(ppc405) {

    ac_format I1 = "%opcd:6 %li:24:s %aa:1 %lk:1";
    ac_format XO1 = "%opcd:6 %rt:5 %ra:5 %rb:5 %oe:1 %xos:9 %rc:1";
    ...

    ac_instr<I1> b;
    ac_instr<XO1> add;
    ...

    ISA_CTOR(ppc405) {

        add.set_asm("add  %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
        add.set_asm("add. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
        add.set_asm("addo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
        add.set_asm("addo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
        add.set_decoder(opcd=31, xos=266);

        b.set_asm("b  %addrRAu", li, aa=0, lk=0);
        b.set_asm("ba %addrRAu", li, aa=1, lk=0);
        b.set_asm("bl %addrRAu", li, aa=0, lk=1);
        b.set_asm("bla %addrRAu", li, aa=1, lk=1);
        b.set_decoder(opcd=18);
        ...

        pseudo_instr("mr %reg, %reg") {
            "or %0, %1, %1";
        }
        ...
    };
};

```

Figura 3.2: Informações de codificação das instruções.

- addo - Soma com indicação de overflow;
- addo. - Soma com atualização do registrador de condição e indicação de overflow.

Note que a instrução *add* é única. Porém a adição dos sufixos citados alteram seus campos *RC* e *OE*, influenciando seu fluxo de execução.

Também no construtor há a indicação de pseudo-instruções, ou seja, aquelas que são válidas na linguagem de montagem, porém não existem no processador. Como é o caso da instrução *mr*, que copia o conteúdo de um registrador em outro. Ela é traduzida em uma instrução *or* que surte o mesmo efeito.

As informações contidas nestes dois primeiros arquivos têm importância especial para o gerador automático de montadores, pois são suficientes para gerar o montador da arquitetura.

No arquivo `ppc405-isa.cpp` descrevemos o comportamento de cada instrução. Aqui pode-se usar todos os recursos que a linguagem SystemC oferece.

Há uma hierarquia de comportamentos que segue a seguinte regra: primeiro é executado o comportamento *instruction*, depois o comportamento referente ao formato da instrução, e só então é executado o comportamento específico da instrução. Isto evita a reescrita da parte do código que é comum a todas as instruções ou a um determinado formato.

Na figura 3.2 há dois exemplos de descrição de comportamento. A instrução *add*, já comentada anteriormente, soma o conteúdo dos registradores-fonte e grava no registrador de destino. Os campos RC e OE indicam se as etapas alternativas serão executadas.

Já a instrução *macchw* foi colocada aí para ilustrar como foram implementadas as instruções da unidade MAC. Elas chamam uma função que tem seu fluxo de execução ditado pelos parâmetros passados. Isso evitou clonagem de código e facilitou na depuração das instruções.

```
#!/Instruction add behavior method.
// Add
void ac_behavior( add )
{
    int result = GPR[ra] + GPR[rb];

    if (oe==1)
        add_XER_OV_SO_update(result,GPR[ra],GPR[rb],0);

    if (rc==1)
        CRO_update(result);

    GPR[rt] = result;
};

#!/Instruction macchw behavior method.
// Multiply Accumulate Cross Halfword to Word Modulo Signed
void ac_behavior( macchw )
{
    genericMac ( NORMAL , CROSS , MODULE , SIGNED , oe , rc , rt , ra , rb);
};
```

Figura 3.3: Descrição do comportamento das instruções *add* e *macchw* no modelo funcional.

O arquivo `ppc405_syscall.cpp` contém a implementação de especializações dos métodos da classe `ac_syscall` que foram feitos para permitir suporte a chamadas de sistema operacional às aplicações, utilizadas, por exemplo, para escrever e ler arquivos no computador

onde está sendo executado o simulador.

O arquivo `ppc405_gdb_funcs.cpp` implementa funções necessárias para se utilizar o modelo com o depurador `gdb`, permitindo escrever e ler registradores e posições na memória.

Todos os arquivos podem ser verificados no final deste documento.

3.3 Modelo com Precisão de Ciclos

O modelo do PowerPC405 com precisão de ciclo foi feito baseado no modelo puramente funcional descrito na seção anterior. As mudanças necessárias para transformar aquele modelo em um que refletisse o tempo necessário para executar a aplicação foram:

- A adição de informações referentes ao pipeline e;
- A descrição do comportamento dividido entre os diferentes estágios do pipeline.

Na figura 3.3 vemos o fragmento de código adicionado ao arquivo `ppc405.ac` que deu origem ao arquivo `ppc405_ca.ac`. Note a declaração dos registradores necessários para o transporte de dados entre os estágios (`ac_reg`), cada um com o seu formato específico (`ac_format`). Há também a declaração do nome dos estágios e sua sequência de execução (`ac_stage`).

```
ac_format F_FE_DE = "%npc:32";

ac_format F_DE_EX = "%npc:32 %data1:32 %data2:32 %data3:32 %ra:5 %rb:5 \
                    %r3:5 %mb:5 %me:5 %sh:5 %nb:5 %u3a:3 %u3b:3 %rf:10 \
                    %regwrite:1 %oe:1 %rc:1 %ui:16 %si:16:s %d:16:s \
                    %ldwrite:1 %stwrite:1 %bo:5 %bi:5 %bd:14:s %aa:1 \
                    %lk:1 %li:24:s";

ac_format F_EX_WB = "%alures:32 %wdata:32 %ea:32 %rdest:5 %regwrite:1 \
                    %stwrite:1 %ldwrite:1 %lddest:5";

ac_format F_WB_LW = "%lddata:32 %lddest:5 %ldwrite:1";

/* FETch, DEcode, EXecute, Write-Back and Load Write-back */
ac_reg<F_FE_DE> FE_DE;
ac_reg<F_DE_EX> DE_EX;
ac_reg<F_EX_WB> EX_WB;
ac_reg<F_WB_LW> WB_LW;

ac_stage FE,DE,EX,WB,LW;
```

Figura 3.4: Descrição dos parâmetros do pipeline.

A divisão da execução entre os estágios deu origem ao novo `ppc405_ca-isa.cpp`, no qual cada comportamento conta com uma sequência de comandos específica de cada estágio. Conforme ilustra a figura 3.3, o comportamento específico da instrução `add` cumpre seu papel somente no estágio EX, pois os outros passos são dados em comportamentos superiores na hierarquia. Assim, o *instruction* cuida da busca no FE, e da escrita no registrador de destino no WB. O XO1 cuida da leitura dos registradores no DE, adiantando-os de outros estágios, se necessário, para evitar perigo de dados.

```
#!/Instruction add behavior method.
void ac_behavior( add )
{
    switch( stage ) {
        case FE: {
            } break;

        case DE: {
            } break;

        case EX: {

            EX_WB.alures = ex_value1 + ex_value2;
            EX_WB.regwrite = DE_EX.regwrite;
            EX_WB.rdest = DE_EX.r3;

            if (DE_EX.oe==1)
                add_XER_OV_SO_update( EX_WB.alures, ex_value1, ex_value2, 0 );
            if (DE_EX.rc==1)
                CRO_update( EX_WB.alures );

            } break;

        case WB: {
            } break;

        case LW: {
            } break;

        default: {
            } break;
        }
    };
};
```

Figura 3.5: Descrição do comportamento da instrução `add` no modelo com precisão de ciclos.

3.4 Dificuldades Encontradas

Uma das partes deste trabalho foi inferir o pipeline do PowerPC, pois não tivemos acesso a documentos que o descrevesse. Os manuais (Xilinx 2003) e (IBM 2005) relatam

a quantidade de estágios e a temporização das instruções em situações específicas.

Na descrição do modelo temporizado levou-se em conta o seguinte:

- FE (Busca) busca a instrução atual e calcula o endereço da próxima;
- DE (Decodificação) lê registradores de propósito geral, adiantados, se houver necessidade, e decodifica os campos no formato da instrução.
- EX (Execução) adianta os valores dos registradores de propósito geral, caso haja necessidade, lê e escreve nos registradores de propósito especial. Em instruções de desvio, o EX calcula o endereço e escreve no PC, invalidando os estágios anteriores. Também calcula o endereço de acesso à memória e executa operações sobre os dados lidos dos registradores.
- WB (Escrita) escreve resultado do EX no banco de registradores, lê e escreve na memória.
- LW (Escrita de carga) escreve o valor lido da memória no banco de registradores.

O problema mais complicado de se resolver foi o caso do desvio incondicional. Inicialmente ele estava sendo feito logo no primeiro estágio, enquanto os desvios condicionais ocorriam no estágio EX, principalmente porque a leitura e escrita de registradores de propósito especial estavam restritas ao estágio EX, e os desvios condicionais os utilizavam. Apesar do fato de invalidar os estágios FE e DE quando ocorria o desvio, eles estavam válidos no mesmo ciclo em que se decidia o desvio, e faziam a busca da instrução. Portanto, se a instrução após o desvio condicional fosse um desvio incondicional, ele buscava a instrução e escrevia no PC. A situação foi contornada adiando-se o desvio que ocorria no FE para o EX. Porém acarretou em invalidar duas instruções a cada desvio, aumentando consideravelmente o tempo de execução. Este aspecto será revisto em uma próxima oportunidade.

O PowerPC405 conta com uma previsão de desvios estática, ou seja, um bit da instrução indica o que fazer. Tal mecanismo ainda não foi implementado. Portanto não há qualquer previsão nos desvios. Isto também será revisto.

4 *Resultados Experimentais*

Este capítulo apresenta a configuração necessária para a reprodução dos resultados aqui listados, a metodologia de validação utilizada, os resultados obtidos na execução dos modelos e a comparação com modelos congêneres.

4.1 Configuração Experimental

O computador utilizado nos experimentos possui um processador Pentium 4 com 3,0 GHz de frequência, cache L2 de 1 MB e memória principal de 1 GB. A execução foi feita no sistema operacional Debian GNU/Linux kernel 2.6. A versão do ArchC utilizado foi a 1.6.0 e a compilação das aplicações do benchmark para executarem no simulador se deu com o compilador redirecionável GCC 3.3.1 disponível no site do projeto ArchC (ArchC 2005). Com a finalidade de gerar os binários para a versão 405 do PowerPC, passamos um parâmetro adicional ao compilador (-mcpu=405).

Para a validação e avaliação de desempenho foi utilizado o MiBench (Guthaus M.), um conhecido benchmark que reflete as aplicações típicas de sistemas embarcados. Ele contém um conjunto de aplicações de seis diferentes segmentos do mercado de embarcados: automotivo, consumidor, rede, escritório, segurança e telecomunicações. Como ilustrado na tabela 4.1. As aplicações desta tabela são, na verdade, um subconjunto do MiBench original, pois nem todas puderam ser executadas, visto que muitas usam threads, e no estágio atual de desenvolvimento do ArchC esta facilidade ainda não pôde ser usada.

Tabela 4.1: Aplicativos do MiBench

Automotivo	Consumidor	Rede	Segurança	Telecomunicações
basicmath	jpeg	dijkstra	rijndael	adpcm
bitcount	lame	patricia	sha	crc32
qsort				fft
susan				gsm

O MiBench (pronuncia-se “*my bench*” em inglês) foi baseado no *EDN Embedded Microprocessor Benchmark Consortium* (EEMBC), que é inviável para pesquisas acadêmicas devido ao alto custo de adesão ao consórcio. O MiBench, por outro lado, é composto de aplicações de código-aberto.

A seguir, uma breve descrição de cada aplicação do MiBench:

1. Automotivo

basicmath executa cálculos matemáticos simples que geralmente não têm suporte de hardware dedicado em processadores embarcados.

bitcount testa a habilidade do processador na manipulação de bits pela contagem do número de bits em um arrays de inteiros.

qsort ordena um array de strings na ordem ascendente usando o conhecido algoritmo “quick sort”.

susan é um pacote de reconhecimento de imagem.

2. Consumidor

jpeg é um formato padrão de compressão de imagens com perda.

lame é um codificador MP3 GPL que suporta codificação com taxa constante, média e variável.

3. Rede

dijkstra constrói um grande grafo representado em uma matriz de adjacências e calcula o menor caminho entre cada par de nodos.

patricia trie é uma estrutura de dados usada no lugar de árvores com nodos-folha muito esparsos. Representam tabelas de roteamento em aplicações de rede.

4. Segurança

rijndael encrypt/decrypt é um cifrador de bloco com a opção de chaves e blocos com 128-, 192-, e 256-bits.

sha é o algoritmo seguro de hash que produz uma mensagem de 160-bit para uma dada entrada.

5. Telecomunicações

adpcm encode/decode é uma representação de som digital que ocupa menos espaço que o PCM.

crc32 é usado para detectar erros em transmissões de dados.

fft/iff executa a transformada rápida de Fourier e sua inversa em um array de dados.

gsm encode/decode é o padrão de codificação/decodificação na Europa e em muitos países. Combina TDMA e FDMA.

Os modelos funcionais do PPC405, do MIPS e do NIOS2 foram executados utilizando um simulador compilado devido ao menor tempo de execução deste sobre o simulador interpretado utilizado nos modelos com precisão de ciclos, pois não suportam a utilização da versão compilada.

4.2 Validação

Na fase de desenvolvimento inicial do modelo foi utilizado um suíte de programas chamado `ac_stone`, disponibilizado pela equipe do ArchC, com a finalidade de avaliar a coerência do modelo na execução de programas simples.

Foi feito um teste preliminar com o `ac_stone` e corrigiu-se erros semânticos mais evidentes. Ele facilitou a identificação da instrução com erro pois possibilitava a depuração do modelo com o `gdb`, uma funcionalidade especial do ArchC. Porém não foi possível fazer o mesmo com o modelo com precisão de ciclos, pois este não provê a possibilidade de uso do depurador.

Já na fase de validação com o MiBench utilizou-se um modelo de referência certificado do ArchC, o modelo do MIPS, considerado assim, confiável para gerar os resultados esperados das aplicações do benchmark. Depois, cada aplicação foi executada no modelo do PowerPC405 para gerar um resultado, que é comparado com o do modelo de referência. Caso houvesse divergência, o modelo era revisado e testado novamente. A figura 4.1 mostra o diagrama da metodologia de validação utilizada. Este esquema foi usado com cada uma das aplicações do MiBench.

4.2.1 Modelo Funcional

A tabela 4.2 lista as aplicações do mibench executadas no modelo funcional do PowerPC405 separadas por segmento. Ela mostra o número de instruções da aplicação, quantas foram

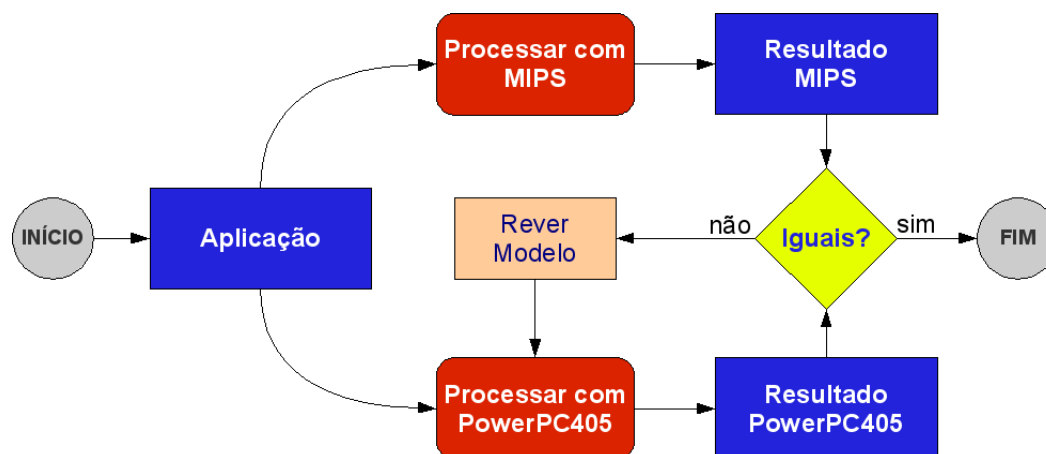


Figura 4.1: Metodologia de Validação

executadas na simulação e o tempo total que o simulador levou para gerar o resultado. Estes dados foram coletados durante a fase de validação e serão usados na próxima seção para comparar o desempenho deste modelo com outros congêneres.

4.2.2 Modelo com Precisão de Ciclos

A tabela 4.5 lista as aplicações do mibench executadas no modelo com precisão de ciclos do PowerPC405 separadas por segmento. Ela mostra o número de instruções da aplicação, quantas foram buscadas na simulação, pois nem todas são executadas, e o tempo total que o simulador levou para gerar o resultado. Estes dados foram coletados durante a fase de validação do modelo com precisão de ciclos e serão usados na próxima seção para comparar o desempenho deste modelo com outros congêneres.

O conjunto de aplicações foi reduzido. Pegou-se os que tomavam menos tempo de simulação pois os modelos com precisão de ciclos demandam muito mais tempo de processamento que os modelos funcionais como veremos na comparação do PPC405 funcional com o PPC405 com precisão de ciclos.

4.3 Comparação de Desempenho

O desempenho é avaliado levando-se em conta o número de instruções gastos pela aplicação e o tamanho que o código irá ocupar na memória, valor que é proporcional ao número de instruções da aplicação, dado que os três modelos utilizam instruções de 4 bytes. Também é considerado o tempo de simulação do modelo na execução de cada aplicação do MiBench.

4.3.1 PPC405 funcional X MIPS funcional X NIOS2 funcional

As aplicações do MiBench da tabela 4.1 foram executados em cada um dos seguintes modelos funcionais:

PowerPC405 Baseado no modelo funcional do PowerPC certificado e disponível no site do projeto ArchC (ArchC 2005), e em um modelo funcional do mesmo processador utilizado no trabalho de conclusão de curso de membros do LAPS, no qual auxiliou em um estudo de caso com relação à geração automática de montadores (Casarotto e Filho 2004). O modelo foi estendido com a implementação das instruções específicas da versão 405, que são as instruções de multiplicação-soma descritas na tabela 2.2.

MIPS Modelo funcional certificado e disponível no site do projeto ArchC (ArchC 2005).

NIOS2 Objeto do trabalho de conclusão de curso de Guilherme Quentel Melo (Melo 2007), membro do LAPS, o mesmo laboratório que viabilizou este trabalho.

A tabela 4.3 lista as aplicações do mibench executadas no modelo funcional do MIPS separadas por segmento. Ela mostra o número de instruções da aplicação, quantas foram executadas na simulação e o tempo total que o simulador levou para gerar o resultado. A tabela 4.4 faz o mesmo para o modelo funcional do NIOS2.

Com relação ao número de instrução de cada aplicação notamos que não há muita entre os processadores considerados, em média o MIPS é praticamente do mesmo tamanho que o do PPC e o do NIOS é 3% maior. Como podemos ver no gráfico 4.2, que mostra o tamanho do programa em número de instruções para as três arquiteturas. Há casos em que o mesmo programa é usado para diferentes fatores de comparação, como é o caso do *susan* que é usado no modo corner, edge e smoothing. Esses programas foram mostrados uma única vez na figura 4.2.

Já no tocante à quantidade de instruções executadas pela aplicação, podemos ver no gráfico 4.3 a porcentagem da quantidade de instruções executadas em relação à quantidade do PPC. O MIPS destaca-se no *gsm*, pois para codificar um dado necessita cerca de 60% a mais de instruções que o PPC, e para decodificar necessita de 20% a menos.

Foi escolhido colocar o gráfico em porcentagens com relação ao PowerPC405, ou seja, normalizado, porque os valores, por benchmark, eram muito discrepantes, e colocá-los

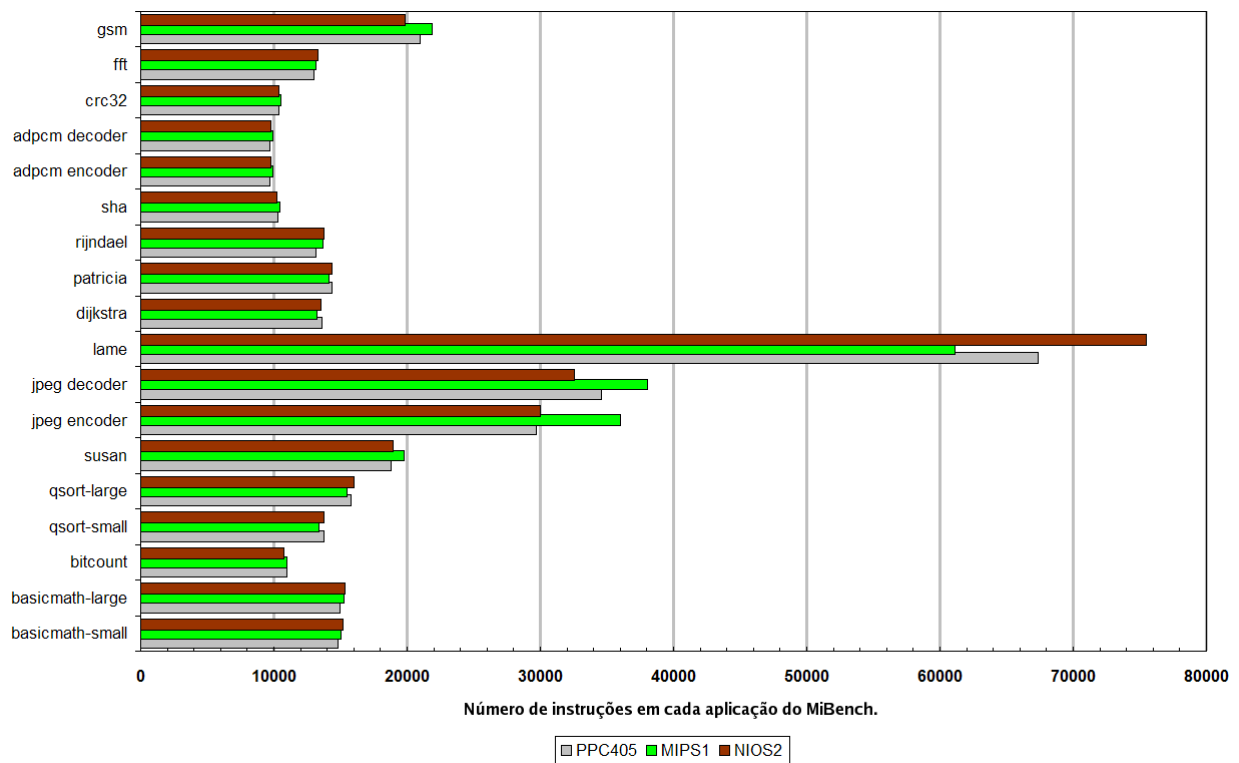


Figura 4.2: Número de instruções de cada benchmark do MiBench nas arquiteturas consideradas.

no mesmo gráfico causaria perda da resolução dos benchmarks que exigem menos instruções. Porém isso nos leva a não poder comparar os benchmarks entre si e sim entre os processadores.

Por fim, o tempo total que os simuladores levaram para gerar os resultados estão ilustrados no gráfico 4.4, que compara as porcentagens em relação ao PowerPC405. Notamos que o MIPS levou um tempo bem maior para ser executado, em média 2,5 vezes o tempo do PowerPC405. O NIOS levou, em média, 50% a mais de tempo.

O gráfico 4.5 nos mostra um resumo das porcentagens dos dados discutidos aqui.

4.3.2 PPC405 ca X MIPS ca X NIOS2 ca

Uma parcela das aplicações do MiBench da tabela 4.1 foram executados em cada um dos seguintes modelos com precisão de ciclos. As mais demoradas foram evitadas por tomarem excessivo tempo de processamento, e as aqui listadas são suficiente para a comparação a que se propõe este trabalho.

PowerPC405ca Baseado no modelo funcional do PowerPC405, este modelo foi confeccionado conforme ilustra o capítulo 2.

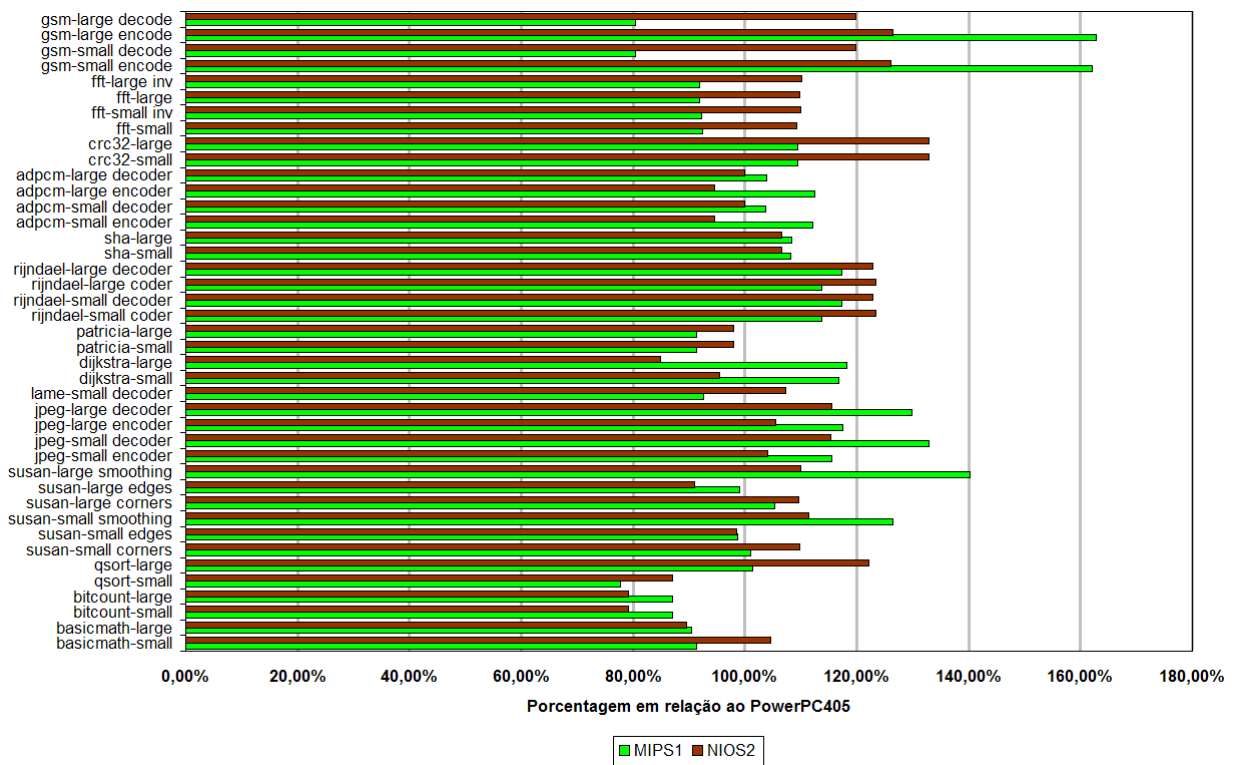


Figura 4.3: Porcentagem de instruções executadas por aplicação do MiBench no modelo funcional do MIPS e do NIOS2 em relação ao modelo funcional do PowerPC405.

MIPSCa Modelo do MIPS R3000 com precisão de ciclos certificado e disponível no site do projeto ArchC (ArchC 2005).

NIOS2ca Também objeto do trabalho de conclusão de curso de Guilherme Quentel Melo (Melo 2007), membro do LAPS, o mesmo laboratório que viabilizou este trabalho.

O número de instruções em cada aplicação não muda conforme o nível de abstração do modelo, tornando a figura 4.2 válida aqui também.

A figura 4.6 mostra a porcentagem da quantidade instruções buscadas em cada aplicação. São chamadas buscadas, e não executadas, porque algumas são descartadas durante a execução, principalmente devido a instruções de desvio. O MIPSCa mantém a quantidade de instruções buscadas no modelo com precisão de ciclos igual à quantidade de instruções executadas no modelo funcional, pois ele faz o desvio no segundo estágio e usa um mecanismo em que a instrução que segue a instrução de desvio deve sempre ser executada, quer ele desvie ou não, isso o torna capaz de não descartar nenhuma instrução buscada.

Nota-se que o MIPSCa mantém o mesmo caso que ocorria na comparação dos modelos funcionais no gsm e que no qsort ele leva metade do tempo do modelo do PowerPC405ca.

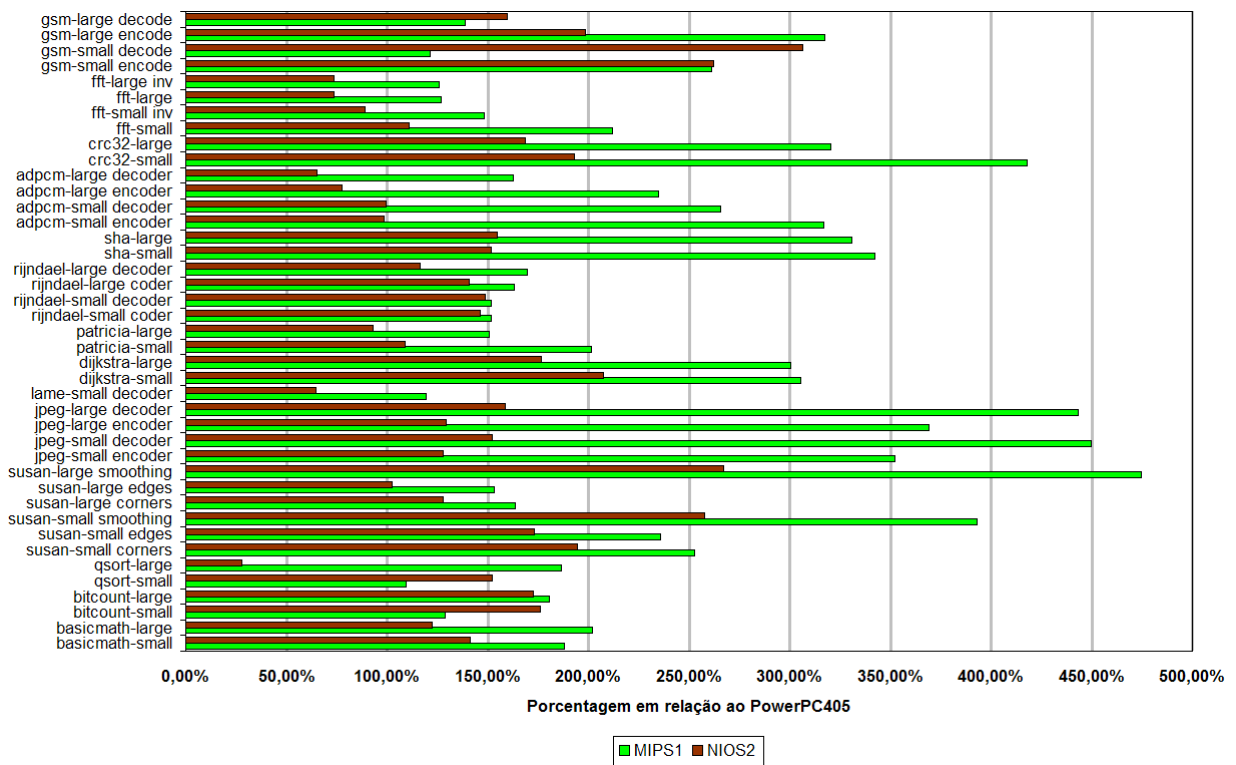


Figura 4.4: Porcentagem do tempo de simulação para gerar os resultados no modelo funcional do MIPS e do NIOS2 em relação aos do modelo funcional do PowerPC405.

O tempo total que o simulador levou para gerar os resultados coloca o modelo do PowerPC405ca em situação de extrema desvantagem. Pois vemos que os outros modelos levaram de 5 a 33% do tempo de simulação que o PowerPC405ca utilizou. O modelo será revisto para sanar eventuais problemas que estejam acarretando esta excessiva demanda de tempo de processamento. As porcentagens estão ilustrados no gráfico 4.7.

O gráfico 4.8 nos leva a um resumo dos valores discutidos nesta comparação. Note que o tempo de simulação dos modelos do MIPSca e do NIOSca necessitaram de 1/5 do tempo que o PowerPC405 levou para gerar os mesmos resultados. Também é notável que o NIOS2ca buscou 13,5% a menos de instruções que o MIPSca e o PowerPC405ca.

4.3.3 PPC405 ca X PPC405 funcional

Aqui comparamos os modelos do processador PowerPC405 nos dois níveis de abstração possíveis no ArchC.

O modelo com precisão de ciclos executa o mesmo número de instruções que o modelo funcional, porém busca de 10 a 50% a mais de instrução. Como mostra a figura 4.9. Isto se deve ao fato de o PowerPC405ca descartar duas instruções a cada instrução de desvio,

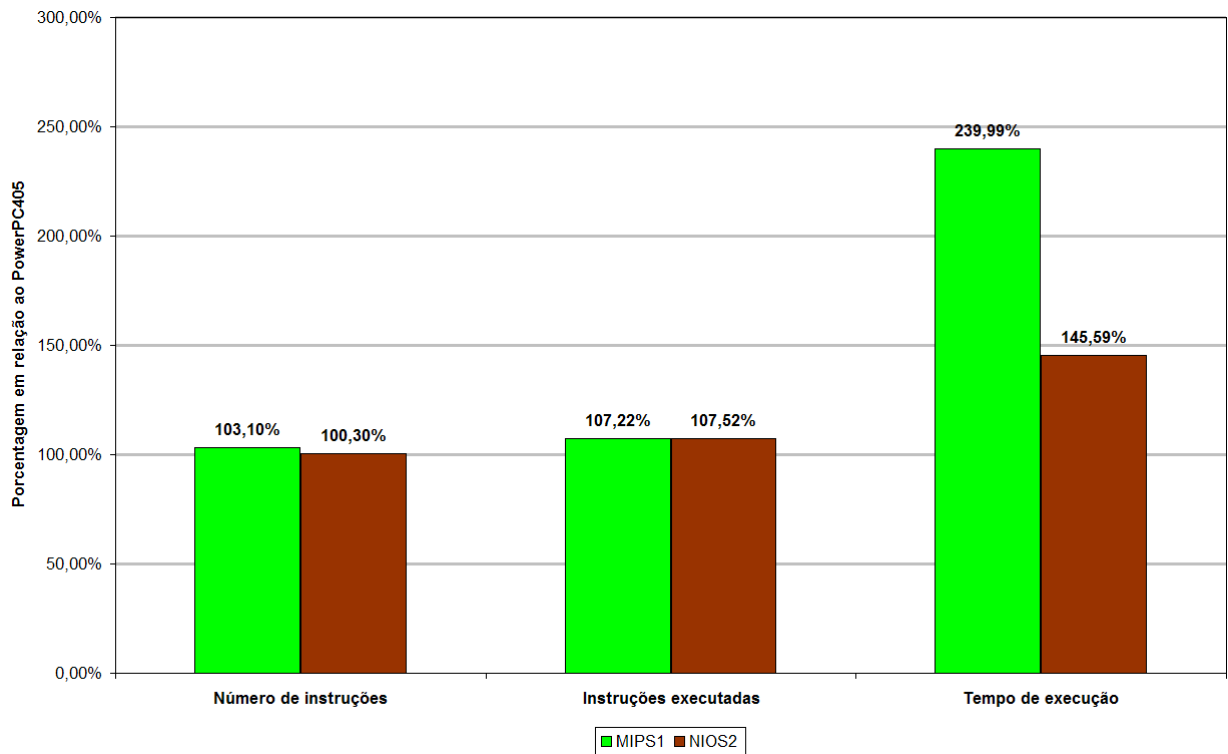


Figura 4.5: Média da porcentagem dos dados coletados do modelo funcional do MIPS e do NIOS2 em relação aos do modelo funcional do PowerPC405.

seja ele tomado ou não.

É interessante notar que sabendo a porcentagem que o PowerPC405ca busca de instruções a mais que o modelo funcional nos dá uma boa ideia de quantas instruções de desvio ele executou. Quanto mais instruções descartadas, mais instruções de desvio foram executadas. Assim, podemos dizer que o adpcm é a aplicação que mais executa instruções de desvio e o rinjdael é o que menos as executa.

A diferença do tempo de simulação do modelo funcional para o modelo com precisão de ciclo é gigantesca. Está ilustrado no gráfico 4.10. Vai de 75.000 a 250.000% a mais de tempo.

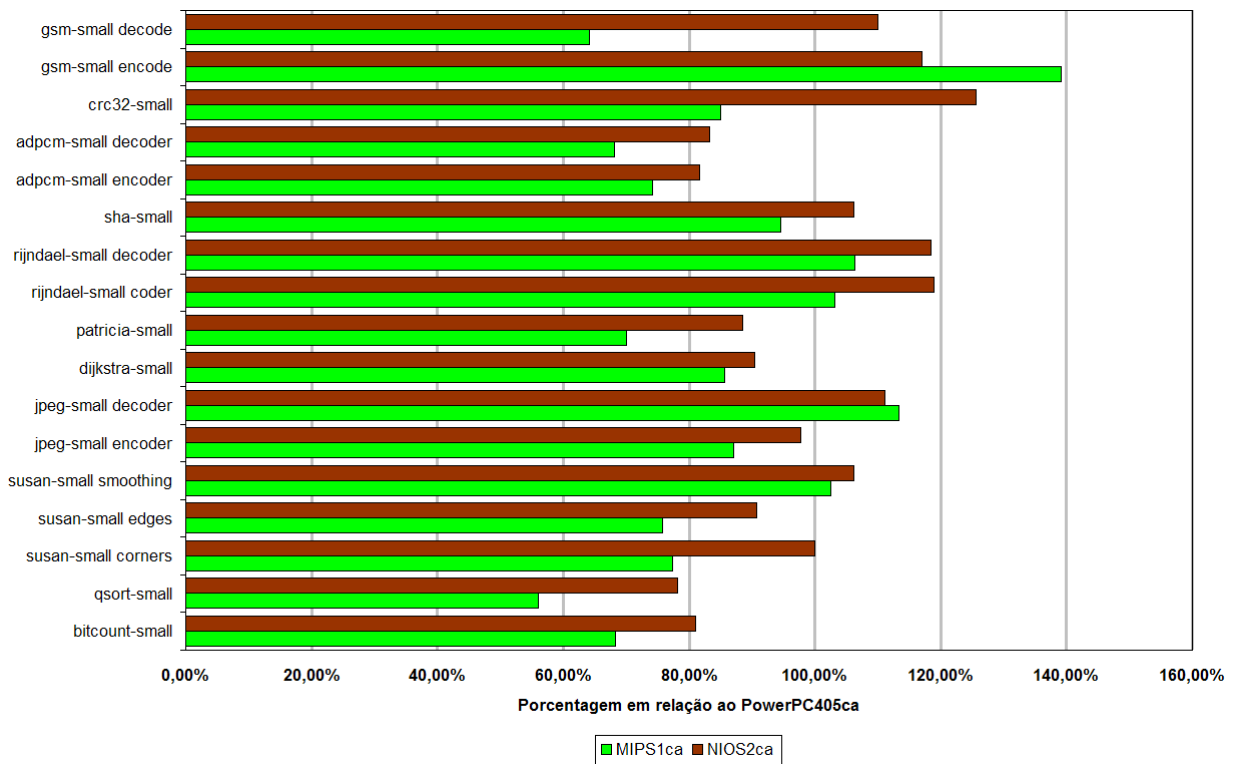


Figura 4.6: Porcentagem de instruções executadas por aplicação do MiBench nos modelos com precisão de ciclos do MIPS e do NIOS2 em relação ao modelo com precisão de ciclos do PowerPC405.

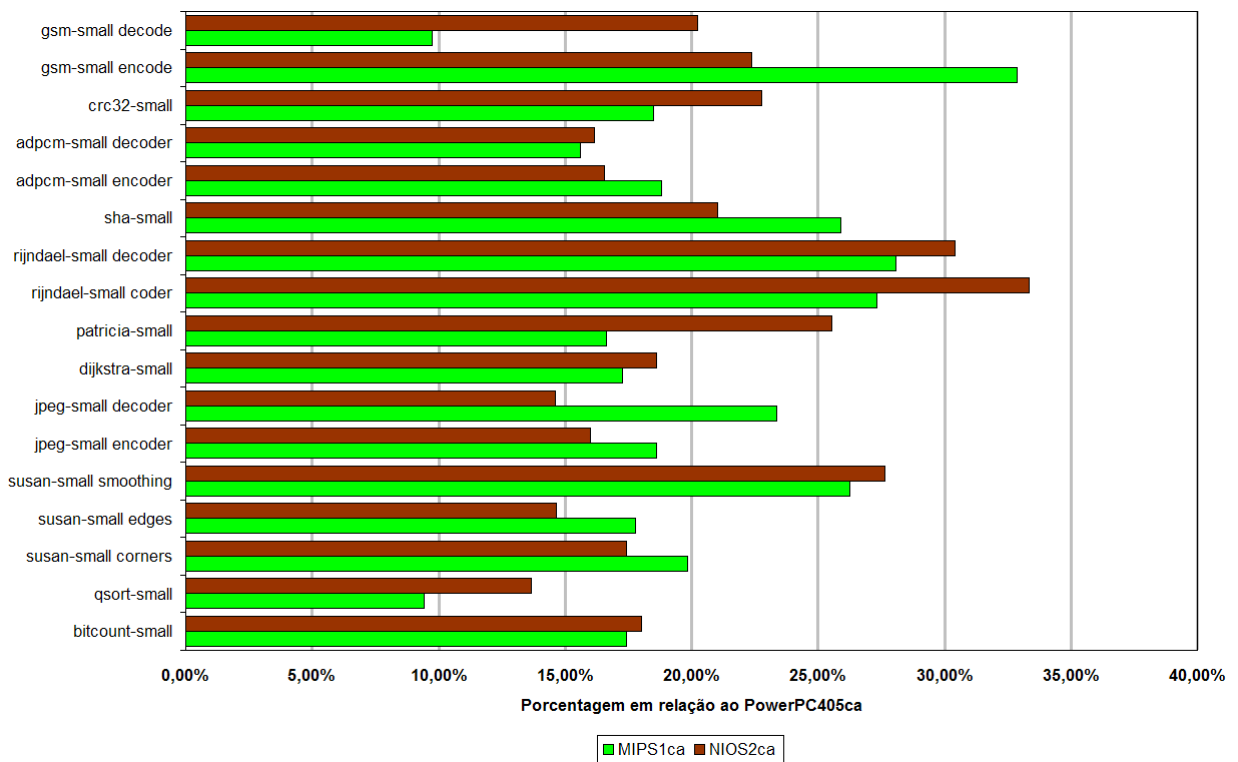


Figura 4.7: Porcentagem do tempo de simulação para gerar os resultados nos modelos com precisão de ciclos do MIPS e do NIOS2 em relação aos do modelo com precisão de ciclos do PowerPC405.

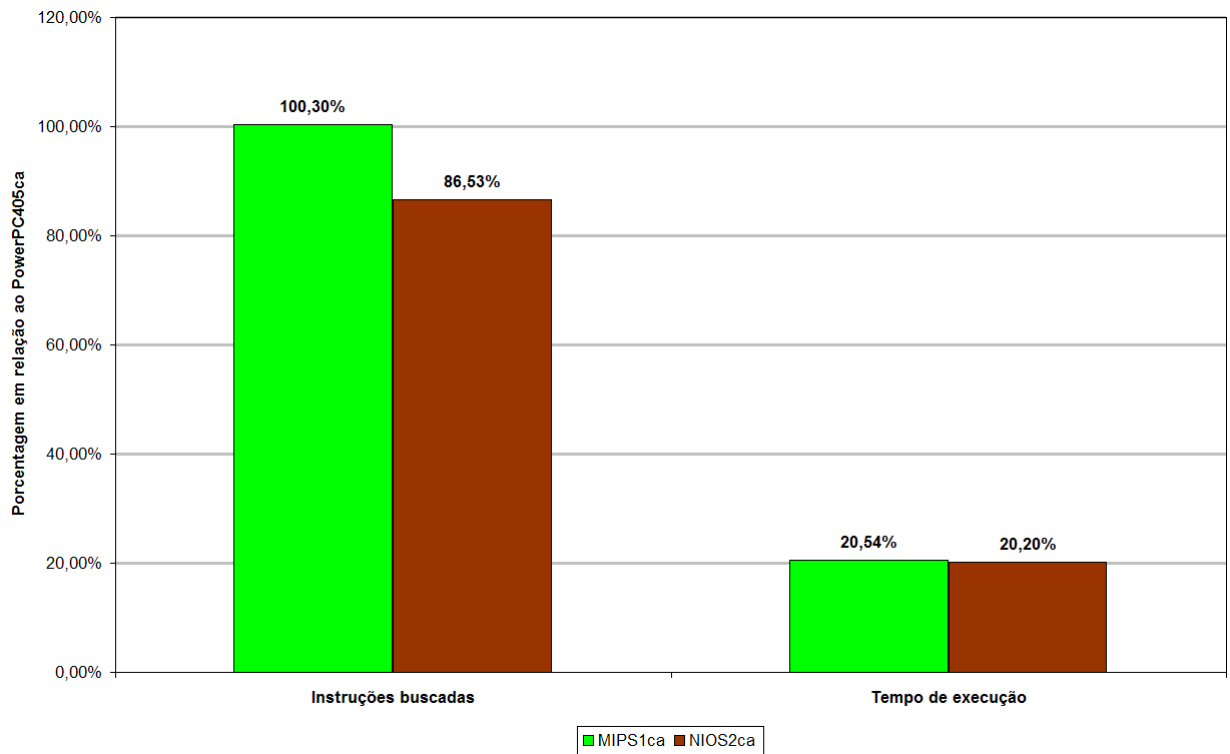


Figura 4.8: Média da porcentagem dos dados coletados dos modelos com precisão de ciclos do MIPS e do NIOS2 em relação aos do modelo com precisão de ciclos do PowerPC405.

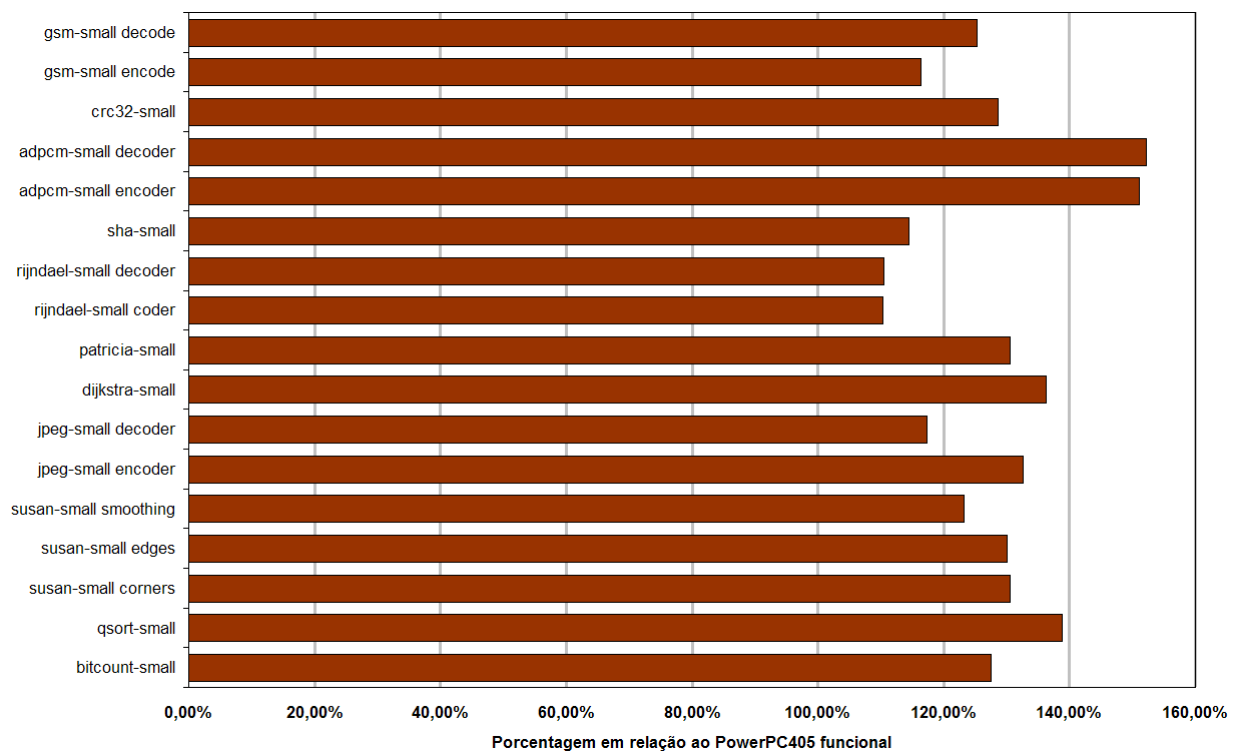


Figura 4.9: Porcentagem de instruções executadas/buscadas por aplicação do MiBench no modelo com precisão de ciclos PowerPC405 em relação ao modelo funcional deste mesmo processador.

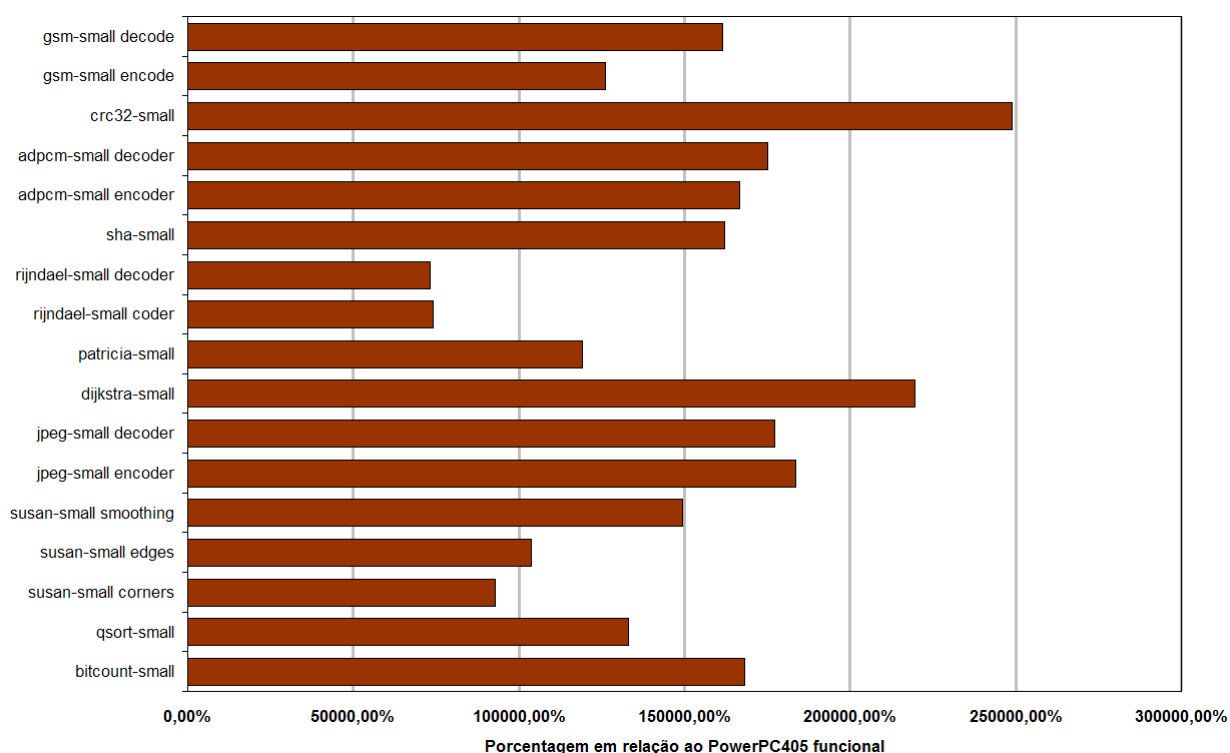


Figura 4.10: Porcentagem do tempo de simulação para gerar os resultados no modelo funcional do MIPS e do NIOS2 em relação aos do modelo funcional do PowerPC405.

Todos os gráficos foram baseados nos dados das tabelas 4.2, 4.3, 4.4, 4.5, 4.6 e 4.7, listadas a seguir:

Tabela 4.2: Conjunto de programas Mibench - PowerPC405 Funcional

Pacote	Programa	Número de instruções	Instruções executadas	Tempo de execução (s)
Automotive	basicmath-small	14818	1491496724	152,0
	basicmath-large	14966	24602904323	3081,1
	bitcount-small	10963	52344934	3,2
	bitcount-large	10963	785501173	48,9
	qsort-small	13795	18548465	1,7
	qsort-large	15817	975192171	110,4
	susan-small corners	18561	3412539	0,4
	susan-small edges	18561	6945490	0,8
	susan-small smoothing	18561	27946125	1,9
	susan-large corners	18561	41846616	5,5
	susan-large edges	18561	177945777	21,2
susan-large smoothing	18561	301874514	18,0	
Consumer	jpeg-small encoder	29735	25513629	1,9
	jpeg-small decoder	34577	6541842	0,5
	jpeg-large encoder	29735	92790698	6,6
	jpeg-large decoder	34577	22619163	1,5
	lame-small decoder	67334	8672094395	1759,5
Network	dijkstra-small	13630	50840786	3,4
	dijkstra-large	13630	241244527	16,7
	patricia-small	14387	316690011	32,1
	patricia-large	14387	2004146016	290,7
Security	rijndael-small coder	13192	29628126	3,6
	rijndael-small decoder	13192	29550668	3,6
	rijndael-large coder	13192	308502242	39,1
	rijndael-large decoder	13192	307693719	48,3
	sha-small	10299	12041205	0,7
	sha-large	10299	125332533	7,1
Telecomm	adpcm-small encoder	9728	30873714	2,3
	adpcm-small decoder	9726	26276530	2,1
	adpcm-large encoder	9728	612472807	63,4
	adpcm-large decoder	9726	518346117	68,6
	crc32-small	10397	28930492	1,5
	crc32-large	10397	562322913	36,9
	fft-small	12985	822828755	100,8
	fft-small inv	12985	1976779878	350,3
	fft-large	12985	16594283924	3387,4
	fft-large inv	12985	16044205614	3347,4
	gsm-small encode	21005	20154916	1,6
	gsm-small decode	21005	11960574	0,8
	gsm-large encode	21005	1082867263	116,3
	gsm-large decode	21005	650858852	77,1

Tabela 4.3: Conjunto de programas Mibench - MIPS Funcional

Pacote	Programa	Número de instruções	Instruções executadas	Tempo de execução (s)
Automotive	basicmath-small	15074	1361274783	285,6
	basicmath-large	15257	22269199867	6219,4
	bitcount-small	11013	45593674	4,1
	bitcount-large	11013	684250382	86,8
	qsort-small	13391	14412623	1,9
	qsort-large	15482	989374482	205,8
	susan-small corners	19794	3458872	0,9
	susan-small edges	19794	6887633	1,8
	susan-small smoothing	19794	35320189	7,5
	susan-large corners	19794	44586473	11,8
	susan-large edges	19794	177422307	45,6
	susan-large smoothing	19794	423392198	85,4
Consumer	jpeg-small encoder	36043	29474823	6,6
	jpeg-small decoder	38042	8697311	2,1
	jpeg-large encoder	36043	109076355	24,3
	jpeg-large decoder	38042	29353751	6,7
	lame-small decoder	61149	8033704502	2102,3
Network	dijkstra-small	13251	59353158	10,4
	dijkstra-large	13251	285280151	50,1
	patricia-small	14108	289205289	64,8
	patricia-large	14108	1830947169	437,8
Security	rijndael-small coder	13712	33715298	5,5
	rijndael-small decoder	13712	34684744	5,5
	rijndael-large coder	13712	351060637	63,7
	rijndael-large decoder	13712	361155494	81,8
	sha-small	10454	13036287	2,3
	sha-large	10454	135696013	23,5
Telecomm	adpcm-small encoder	9899	34628836	7,4
	adpcm-small decoder	9893	27256674	5,6
	adpcm-large encoder	9899	688972768	148,7
	adpcm-large decoder	9893	538659721	111,5
	crc32-small	10526	31642815	6,1
	crc32-large	10526	615051603	118,3
	fft-small	13174	760568629	213,3
	fft-small inv	13174	1822953522	519,2
	fft-large	13174	15244218367	4301,5
	fft-large inv	13174	14750402915	4210,7
	gsm-small encode	21900	32662868	4,3
	gsm-small decode	21900	9613511	0,9
	gsm-large encode	21900	1763291205	369,0
	gsm-large decode	21900	523197240	107,0

Tabela 4.4: Conjunto de programas Mibench - NIOS2 Funcional

Pacote	Programa	Número de instruções	Instruções executadas	Tempo de execução (s)
Automotive	basicmath-small	15191	1561063336	215,0
	basicmath-large	15369	22039599769	3767,0
	bitcount-small	10763	41479000	5,6
	bitcount-large	10763	622394114	83,0
	qsort-small	13745	16136424	2,6
	qsort-large	15995	192013514	31,0
	susan-small corners	18912	3759468	0,7
	susan-small edges	18912	6880985	1,3
	susan-small smoothing	18912	31118438	4,9
	susan-large corners	18912	46385816	9,2
	susan-large edges	18912	163131547	30,5
	susan-large smoothing	18912	332160363	48,0
Consumer	jpeg-small encoder	29970	26552292	2,4
	jpeg-small decoder	32576	7542980	0,7
	jpeg-large encoder	29970	97888608	8,5
	jpeg-large decoder	32576	26130691	2,4
	lame-small decoder	75524	9306508888	1138,7
Network	dijkstra-small	13567	48535523	7,1
	dijkstra-large	13567	204657540	29,4
	patricia-small	14394	309980718	35,0
	patricia-large	14394	1964724183	270,0
Security	rijndael-small coder	13776	36561506	5,3
	rijndael-small decoder	13776	36289080	5,4
	rijndael-large coder	13776	380709693	54,9
	rijndael-large decoder	13776	377868505	56,1
	sha-small	10256	12834645	1,0
	sha-large	10256	133605127	11,0
Telecomm	adpcm-small encoder	9747	29179525	2,3
	adpcm-small decoder	9741	26270157	2,1
	adpcm-large encoder	9747	579119635	49,2
	adpcm-large decoder	9741	518201806	44,6
	crc32-small	10369	38469170	2,8
	crc32-large	10369	747721150	62,3
	fft-small	13280	898972685	112,0
	fft-small inv	13280	2174894392	312,0
	fft-large	13280	18227603026	2490,5
	fft-large inv	13280	17682569892	2468,9
	gsm-small encode	19847	25404166	4,3
	gsm-small decode	19847	14326004	2,3
	gsm-large encode	19847	1369426957	231,0
	gsm-large decode	19847	779962071	123,0

Tabela 4.5: Conjunto de programas Mibench - PowerPC405 com Precisão de Ciclos

Pacote	Programa	Número de instruções	Instruções buscadas	Tempo de execução (s)
Automotive	bitcount-small	10963	66774644	5347,6
	qsort-small	13795	25735539	2274,7
	susan-small corners	18767	4468109	333,9
	susan-small edges	18767	9086674	778,8
	susan-small smoothing	18767	34424177	2841,1
Consumer	jpeg-small encoder	29735	33838859	3448,9
	jpeg-small decoder	34577	7673592	815,7
Network	dijkstra-small	13630	69257416	7513,3
	patricia-small	14387	413266071	38265,9
Security	rijndael-small coder	13192	32665360	2676,8
	rijndael-small decoder	13192	32627232	2660,3
	sha-small	10299	13786971	1069,2
Telecomm	adpcm-small encoder	9728	46631970	3902,5
	adpcm-small decoder	9726	39992072	3693,0
	crc32-small	10397	37209932	3609,5
	gsm-small encode	21005	23462280	2065,8
	gsm-small decode	21005	14991282	1212,3

Tabela 4.6: Conjunto de programas Mibench - MIPS com Precisão de Ciclos

Pacote	Programa	Número de instruções	Instruções buscadas	Tempo de execução (s)
Automotive	bitcount-small	11013	45593674	931,7
	qsort-small	13391	14412623	214,6
	susan-small corners	19794	3458872	66,2
	susan-small edges	19794	6887633	138,4
	susan-small smoothing	19794	35320189	745,7
Consumer	jpeg-small encoder	36043	29474823	642,2
	jpeg-small decoder	38042	8697311	190,5
Network	dijkstra-small	13251	59353158	1298,4
	patricia-small	14108	289205289	6365,8
Security	rijndael-small coder	13712	33715298	731,0
	rijndael-small decoder	13712	34684744	746,5
	sha-small	10454	13036287	276,8
Telecomm	adpcm-small encoder	9899	34628836	734,6
	adpcm-small decoder	9893	27256674	575,8
	crc32-small	10526	31642891	667,8
	gsm-small encode	21900	32662613	679,1
	gsm-small decode	21900	9613256	117,9

Tabela 4.7: Conjunto de programas Mibench - NIOS2 com Precisão de Ciclos

Pacote	Programa	Número de instruções	Instruções buscadas	Tempo de execução (s)
Automotive	bitcount-small	10763	54108933	964,1
	qsort-small	13745	20118742	311,1
	susan-small corners	18912	4465279	58,2
	susan-small edges	18912	8252388	114,1
	susan-small smoothing	18912	36563256	785,7
Consumer	jpeg-small encoder	29970	33096950	551,4
	jpeg-small decoder	32576	8522363	119,3
Network	dijkstra-small	13567	62641949	1398,4
	patricia-small	14394	366072907	9778,4
Security	rijndael-small coder	13776	38842459	892,4
	rijndael-small decoder	13776	38646011	809,6
	sha-small	10256	14650038	225,0
Telecomm	adpcm-small encoder	9747	38079617	646,0
	adpcm-small decoder	9741	33269427	596,4
	crc32-small	10369	46722296	822,4
	gsm-small encode	19847	27451209	462,3
	gsm-small decode	19847	16481579	245,4

5 Conclusão

Neste trabalho extendemos o modelo puramente funcional do PowerPC disponível no site do Projeto ArchC, adicionando-lhe as instruções específicas da Unidade de Multiplicação-Soma do PowerPC405. Também descrevemos o modelo temporizado deste processador com a definição do seu pipeline e a divisão do fluxo de execução de cada instrução em estágios. Foram executadas aplicações típicas de sistemas embarcados nos modelos aqui descritos e em quatro outros congêneres, MIPS funcional, NiosII funcional, MIPS com precisão de ciclos e NiosII com precisão de ciclos.

O modelo puramente funcional apresentou resultados próximos dos outros funcionais. Porém o modelo com precisão de ciclos, apesar de executar corretamente as aplicações apresentou-se lento na simulação e necessitou de 1/3 a mais de busca de instruções devido a um fator já discutido no capítulo anterior.

Há apenas dois modelos de processadores com precisão de ciclos disponíveis no Projeto ArchC. Um é do MIPS R3000, que usamos aqui, e o outro do NiosII, que também usamos, e ainda será certificado. Portanto há uma certa escassez de modelos com esta característica para serem usados na exploração de arquiteturas e na validação de ferramentas por parte dos desenvolvedores da linguagem ArchC. Com o modelo temporizado aqui descrito, espera-se ter contribuído com mais uma alternativa de utilização desse nível de abstração. Lembrando que o modelo será revisto a fim de torná-lo mais eficiente com relação ao tempo de simulação e o número de instruções buscadas.

5.1 Produtos Gerados

- Modelo do PowerPC405 puramente funcional
- Modelo do PowerPC405 com precisão de ciclos
- Artigo a ser submetido ao *Students Forum on Microelectronics 2007*

- Artigo a ser submetido ao Iberchip 2008

5.2 Trabalhos Futuros

Aqui procuramos listar as idéias de pesquisas futuras vislumbradas durante a realização deste trabalho:

- Implementação do mecanismo de previsão estática presente no PowerPC405;
- Fazer a divisão de leitura, modificação e escrita em diferentes estágios para os registradores de propósito especial como é feito aqui para os de propósito geral. Isto exigirá um mecanismo de adiantamento dos mesmos;
- Fazer a divisão de leitura, modificação e escrita em diferentes estágios para as instruções de acesso múltiplo como: carga e escrita de múltiplas palavras em diferentes registradores, bem como a carga e escrita de *strings*;
- Estudar uma forma de garantir execução paralela de várias unidades funcionais no estágio EX do pipeline;
- Construção de uma árvore de modelos de processadores de uma mesma família possibilitando o reuso de código;
- Geração automática de modelos com precisão de ciclos partindo-se de modelos puramente funcionais;
- Partindo-se de modelos ArchC temporizados, estudar a possibilidade de geração do código SystemC no Nível de Transferência de Registradores (RTL) possibilitando sua síntese, direta ou indireta(VHDL/Verilog), para teste em FPGA, ou até mesmo a síntese do hardware;
- Adequação ao padrão IEEE 1666 do código do simulador SystemC gerado pelo ArchC.

Referências

- ArchC 2005 ARCHC. *The ArchC Architecture Description Language*. v1.5. [S.l.], JUL 2005. Disponível em: <<http://www.archc.or>>. 12, 23, 24, 31, 35, 37
- Bhasker 2002 BHASKER, J. *A SystemC Primer*. [S.l.]: Star Galaxy Publishing, 2002. 23
- Casarotto e Filho 2004 CASAROTTO, D. C.; FILHO, J. O. C. *Geração Automática de Montadores a Partir de ArchC: Um Estudo de Caso com o PowerPC 405*. Universidade Federal de Santa Catarina, 2004. Trabalho de Conclusão de Curso. 24, 35
- EEMBC EEMBC. *EDN Embedded Microprocessor Benchmark Consortium*. Disponível em: <<http://www.eembc.or>>. 32
- Guthaus M. GUTHAUS M., e. a. Mibench: A free, commercially representative embedded benchmark suite. *IEEE 4th Annual Workshop on Workload Characterization*. Disponível em: <<http://www.eecs.umich.edu/mibench>>. 31
- Hennessy e Patterson 2000 HENNESSY, J. L.; PATTERSON, D. A. *Organização e Projeto de Computadores: A Interface Hardware/Software*. 2. ed. [S.l.]: Editora LTC, 2000. 21
- IBM 2005 IBM. *PPC405Fx Embedded Processor Core User's Manual*. [S.l.], JAN 2005. Disponível em: <<http://www.ibm.co>>. 13, 15, 29
- Keutzer et al. 2000 KEUTZER, K. et al. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, DEZ 2000. 11
- Melo 2007 MELO, G. Q. *Modelagem do Processador NIOS2 para uma Plataforma de SoCs*. Universidade Federal de Santa Catarina, 2007. Trabalho de Conclusão de Curso. 35, 37
- Moore 1965 MOORE, G. E. Cramming more components onto integrated circuits. 1965. Disponível em: <<http://download.intel.com/research/silicon/moorespaper.pd>>. 11
- Rigo et al. 2004 RIGO, S. et al. Teaching computer architecture using an architecture description language. *Workshop on Computer Architecture Education*, p. 22–28, 2004. 23, 24
- Sangiovanni-Vincentelli e Martin 2001 SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, NOV 2001. 11

SystemC SYSTEMC. *The Open SystemC Initiative*. Disponível em: <[http://www-systemc.or](http://www.systemc.or)>. 11, 23

Xilinx 2003 XILINX. *PowerPC Processor Reference Guide - Embedded Development Kit*. [S.l.], SET 2003. Disponível em: <<http://www.xilinx.co>>. 13, 29

Anexo A - Código Fonte

5.1 Funcional

5.1.1 ppc405.ac

```

/*****/
/* The ArchC PowerPC405 functional model.          */
/* Author: Bruno Corsi dos Santos                 */
/*      Alexandro Baldassin (assembly information) */
/*      Sandro Carvalho (PowerPC405 instructions) */
/*                                                    */
/* For more information on ArchC, please visit:    */
/* http://www.archc.org                            */
/*                                                    */
/* The ArchC Team                                  */
/* Computer Systems Laboratory (LSC)                */
/* IC-UNICAMP                                       */
/* http://www.lsc.ic.unicamp.br                    */
/*                                                    */
/* System Design Automation Lab (LAPS)             */
/* INF-UFSC                                         */
/* http://www.laps.inf.ufsc.br                     */
/*****/

```

```
AC_ARCH(ppc405) {
```

```
    ac_wordsize 32;
```

```
    ac_mem MEM:8M;
```

```
    ac_regbank GPR:32;
```

```
ac_reg SPRG4;
ac_reg SPRG5;
ac_reg SPRG6;
ac_reg SPRG7;
ac_reg USPRG0;

ac_reg XER;

ac_reg MSR;

ac_reg EVPR;
ac_reg SRR0;
ac_reg SRR1;

ac_reg CR;
ac_reg LR;
ac_reg CTR;

ARCH_CTOR(ppc405) {
    ac_isa("ppc405_isa.ac");
    set_endian("big");
};

};
```

5.1.2 ppc405_isa.ac

```
/*
/*****
/* The ArchC PowerPC405 functional model.
/* Instruction Set Architecture Description
/* Author: Bruno Corsi dos Santos
/* Alexandre Baldassin (assembly information)
/* Sandro Carvalho (PowerPC405 instructions)
/*
```

```

/* For more information on ArchC, please visit:      */
/* http://www.archc.org                             */
/*                                                  */
/* The ArchC Team                                    */
/* Computer Systems Laboratory (LSC)                 */
/* IC-UNICAMP                                        */
/* http://www.lsc.ic.unicamp.br                     */
/*                                                  */
/* System Design Automation Lab (LAPS)               */
/* INF-UFSC                                          */
/* http://www.laps.inf.ufsc.br                      */
/*****

```

```
AC_ISA(ppc405) {
```

```
    ac_format I1 = "%opcd:6 %li:24:s %aa:1 %lk:1";
```

```
    ac_format B1 = "%opcd:6 %bo:5 %bi:5 %bd:14:s %aa:1 %lk:1";
```

```
    ac_format SC1 = "%opcd:6 0x00:5 0x00:5 0x00:4 %lev:7 0x00:3 0x01:1 0x00:1";
```

```
    ac_format D1 = "%opcd:6 %rt:5 %ra:5 %d:16:s";
```

```
    ac_format D2 = "%opcd:6 %rs:5 %ra:5 %si:16:s";
```

```
    ac_format D3 = "%opcd:6 %rs:5 %ra:5 %d:16:s";
```

```
    ac_format D4 = "%opcd:6 %rs:5 %ra:5 %ui:16";
```

```
    ac_format D5 = "%opcd:6 %bf:3 0x00:1 %l:1 %ra:5 %si:16:s";
```

```
    ac_format D6 = "%opcd:6 %bf:3 0x00:1 %l:1 %ra:5 %ui:16";
```

```
    ac_format D7 = "%opcd:6 %to:5 %ra:5 %si:16:s";
```

```
    ac_format X1 = "%opcd:6 %rt:5 %ra:5 %rb:5 %xog:10 %rc:1";
```

```
    ac_format X2 = "%opcd:6 %rt:5 %ra:5 %rb:5 %xog:10 0x00:1";
```

```
    ac_format X3 = "%opcd:6 %rt:5 %ra:5 %nb:5 %xog:10 0x00:1";
```

```
    ac_format X4 = "%opcd:6 %rt:5 %ra:5 %ws:5 %xog:10 0x00:1";
```

```
    ac_format X5 = "%opcd:6 %rt:5 0x00:5 %rb:5 %xog:10 0x00:1";
```

```
    ac_format X6 = "%opcd:6 %rt:5 0x00:5 0x00:5 %xog:10 0x00:1";
```



```

ac_format X7 = "%opcd:6 %rs:5 %ra:5 %rb:5 %xog:10 %rc:1";
ac_format X8 = "%opcd:6 %rs:5 %ra:5 %rb:5 %xog:10 0x01:1";
ac_format X9 = "%opcd:6 %rs:5 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X10 = "%opcd:6 %rs:5 %ra:5 %nb:5 %xog:10 0x00:1";
ac_format X11 = "%opcd:6 %rs:5 %ra:5 %ws:5 %xog:10 0x00:1";
ac_format X12 = "%opcd:6 %rs:5 %ra:5 %sh:5 %xog:10 %rc:1";
ac_format X13 = "%opcd:6 %rs:5 %ra:5 0x00:5 %xog:10 %rc:1";
ac_format X14 = "%opcd:6 %rs:5 0x00:5 %rb:5 %xog:10 0x00:1";
ac_format X15 = "%opcd:6 %rs:5 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X16 = "%opcd:6 %bf:3 0x00:1 %l:1 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X17 = "%opcd:6 %bf:3 0x00:2 %bfa:3 0x00:2 0x00:5 %xog:10 %rc:1";
ac_format X18 = "%opcd:6 %bf:3 0x00:2 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X19 = "%opcd:6 %bf:3 0x00:2 0x00:5 %u:5 %xog:10 %rc:1";
ac_format X20 = "%opcd:6 %bf:3 0x00:2 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X21 = "%opcd:6 %to:5 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X22 = "%opcd:6 %bt:5 0x00:5 0x00:5 %xog:10 %rc:1";
ac_format X23 = "%opcd:6 0x00:5 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X24 = "%opcd:6 0x00:5 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X25 = "%opcd:6 0x00:5 0x00:5 %e:1 0x00:4 %xog:10 0x00:1";

ac_format XL1 = "%opcd:6 %bt:5 %ba:5 %bb:5 %xog:10 0x00:1";
ac_format XL2 = "%opcd:6 %bo:5 %bi:5 0x00:3 %bh:2 %xog:10 %lk:1";
ac_format XL3 = "%opcd:6 %bf:3 0x00:2 %bfa:3 0x00:2 0x00:5 %xog:10 0x00:1";
ac_format XL4 = "%opcd:6 0x00:5 0x00:5 0x00:5 %xog:10 0x00:1";

ac_format XFX1 = "%opcd:6 %rt:5 %sprf:10 %xog:10 0x00:1";
ac_format XFX2 = "%opcd:6 %rt:5 %dcrf:10 %xog:10 0x00:1";
ac_format XFX3 = "%opcd:6 %rs:5 0x00:1 %xfm:8 0x00:1 %xog:10 0x00:1";
ac_format XFX4 = "%opcd:6 %rs:5 %sprf:10 %xog:10 0x00:1";
ac_format XFX5 = "%opcd:6 %rs:5 %dcrf:10 %xog:10 0x00:1";

ac_format X01 = "%opcd:6 %rt:5 %ra:5 %rb:5 %oe:1 %xos:9 %rc:1";
ac_format X02 = "%opcd:6 %rt:5 %ra:5 %rb:5 0x00:1 %xos:9 %rc:1";
ac_format X03 = "%opcd:6 %rt:5 %ra:5 0x00:5 %oe:1 %xos:9 %rc:1";

```

```
ac_format M1 = "%opcd:6 %rs:5 %ra:5 %rb:5 %mb:5 %me:5 %rc:1";
ac_format M2 = "%opcd:6 %rs:5 %ra:5 %sh:5 %mb:5 %me:5 %rc:1";

ac_instr<I1> b;

ac_instr<B1> bc;

ac_instr<SC1> sc;

ac_instr<D1> addi, addic, addic_, addis, lbz, lbzu, lha, lhau, lhz,
    lhzu, lmw, lwz, lwzu, mulli, subfic;

ac_instr<D3> stb, stbu, sth, sthu, stmw, stw, stwu;

ac_instr<D4> andi_, andis_, ori, oris, xori, xoris;

ac_instr<D5> cmpi;

ac_instr<D6> cmpli;

ac_instr<X1> mulchw, mulchwu, mulhhw, mulhhwu,
    mullhw, mullhwu;

ac_instr<X2> lbzux, lbzx, lhaux, lhax, lhbrx, lhzux, lhzx, lswx,
    lwbrx, lwzux, lwzx;

ac_instr<X3> lswi;

ac_instr<X6> mfcr;

ac_instr<X7> ande, andc, eqv, nand,
    nor, ore, orc, slw, sraw,
    srw, xxor;

ac_instr<X9> stbux, stbx, sthbrx, sthux, stswx, stwbrx, stwux, stwx,
```

```
    sthx;
```

`ac_instr<X10>` stswi;

`ac_instr<X12>` srawi;

`ac_instr<X13>` cntlzw, extsb, extsh;

`ac_instr<X16>` cmp, cmpl;

`ac_instr<X18>` mcrxr;

`ac_instr<XL1>` crand, crandc, creqv, crnand, crnor, cror, crorc, crxor;

`ac_instr<XL2>` bcctr, bclr;

`ac_instr<XL3>` mcrf;

`ac_instr<XFX1>` mfspr;

`ac_instr<XFX3>` mtrcf;

`ac_instr<XFX4>` mtspr;

`ac_instr<X01>` add, addc, adde, mullw, divw, divwu,
 subf, subfc, subfe;

`ac_instr<X01>` macchw, macchws, macchwsu, macchwu,
 machhw, machhws, machhwsu, machhwu,
 maclhw, maclhws, maclhwsu, maclhwu,
 nmacchw, nmacchws, nmachhw, nmachhws,
 nmaclhw, nmaclhws;

`ac_instr<X02>` mulhw, mulhwu;

```
ac_instr<X03> addme, addze, neg, subfme, subfze;
```

```
ac_instr<M1> rlwnm;
```

```
ac_instr<M2> rlwimi, rlwinm;
```

```
ac_asm_map reg {
    ""[0..31] = [0..31];
}
```

```
ISA_CTOR(ppc405) {
```

```
    add.set_asm("add  %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
    add.set_asm("add. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
    add.set_asm("addo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
    add.set_asm("addo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
    add.set_decoder(opcd=31, xos=266);
```

```
    addc.set_asm("addc %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
    addc.set_asm("addc. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
    addc.set_asm("addco %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
    addc.set_asm("addco. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
    addc.set_decoder(opcd=31, xos=10);
```

```
    adde.set_asm("adde %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
    adde.set_asm("adde. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
    adde.set_asm("addeo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
    adde.set_asm("addeo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
    adde.set_decoder(opcd=31, xos=138);
```

```
    addi.set_asm("li %reg, %exp", rt, ra=0, d);
    addi.set_asm("li %reg, %expHc@ha", rt, ra=0, d);
    addi.set_asm("la %reg, %exp@l(%imm)", rt, d, ra );
    addi.set_asm("addi %reg, %reg, %exp", rt, ra, d);
    addi.set_decoder(opcd=14);
```

```
addic.set_asm("addic %reg, %reg, %exp", rt, ra, d);
addic.set_decoder(opcd=12);

addic_.set_asm("addic. %reg, %reg, %exp", rt, ra, d);
addic_.set_decoder(opcd=13);

addis.set_asm("lis %reg, %exp", rt, ra=0, d);
addis.set_asm("lis %reg, %expHc@ha", rt, ra=0, d);
addis.set_asm("addis %reg, %reg, %exp", rt, ra, d);
addis.set_decoder(opcd=15);

addme.set_asm("addme %reg, %reg", rt, ra, oe=0, rc=0);
addme.set_asm("addme. %reg, %reg", rt, ra, oe=0, rc=1);
addme.set_asm("addmeo %reg, %reg", rt, ra, oe=1, rc=0);
addme.set_asm("addmeo. %reg, %reg", rt, ra, oe=1, rc=1);
addme.set_decoder(opcd=31, xos=234);

addze.set_asm("addze %reg, %reg", rt, ra, oe=0, rc=0);
addze.set_asm("addze. %reg, %reg", rt, ra, oe=0, rc=1);
addze.set_asm("addzeo %reg, %reg", rt, ra, oe=1, rc=0);
addze.set_asm("addzeo. %reg, %reg", rt, ra, oe=1, rc=1);
addze.set_decoder(opcd=31, xos=202);

ande.set_asm("and %reg, %reg, %reg", ra, rs, rb, rc=0);
ande.set_asm("and. %reg, %reg, %reg", ra, rs, rb, rc=1);
ande.set_decoder(opcd=31, xog=28);

andc.set_asm("andc %reg, %reg, %reg", ra, rs, rb, rc=0);
andc.set_asm("andc. %reg, %reg, %reg", ra, rs, rb, rc=1);
andc.set_decoder(opcd=31, xog=60);

andi_.set_asm("andi. %reg, %reg, %imm", ra, rs, ui);
andi_.set_decoder(opcd=28);
```

```
andis_.set_asm("andis. %reg, %reg, %imm", ra, rs, ui);
andis_.set_decoder(opcd=29);

b.set_asm("b %addrRAu", li, aa=0, lk=0);
b.set_asm("ba %addrRAu", li, aa=1, lk=0);
b.set_asm("bl %addrRAu", li, aa=0, lk=1);
b.set_asm("bla %addrRAu", li, aa=1, lk=1);
b.set_decoder(opcd=18);

bc.set_asm("bc %imm, %exp, %addrRAu", bo, bi, bd, aa=0, lk=0);
bc.set_asm("bca %imm, %exp, %addrRAu", bo, bi, bd, aa=1, lk=0);
bc.set_asm("bcl %imm, %exp, %addrRAu", bo, bi, bd, aa=0, lk=1);
bc.set_asm("bcla %imm, %exp, %addrRAu", bo, bi, bd, aa=1, lk=1);
bc.set_decoder(opcd=16);

bcctr.set_asm("bcctr", bo=0x14, bi=0, bh=0, lk=0);
bcctr.set_asm("bcctl", bo=0x14, bi=0, bh=0, lk=1);
bcctr.set_asm("bcctr %reg, %reg, %reg", bo, bi, bh, lk=0);
bcctr.set_asm("bcctl %reg, %reg, %reg", bo, bi, bh, lk=1);
bcctr.set_decoder(opcd=19, xog=528);

bclr.set_asm("blr", bo=0x14, bi=0, bh=0, lk=0);
bclr.set_asm("bclr %reg, %reg, %reg", bo, bi, bh, lk=0);
bclr.set_asm("bclrl %reg, %reg, %reg", bo, bi, bh, lk=1);
bclr.set_decoder(opcd=19, xog=16);

cmp.set_asm("cmpw %imm, %imm, %imm", bf, ra, rb, l=0);
cmp.set_asm("cmp %imm, %imm, %reg, %reg", bf, l, ra, rb);
cmp.set_decoder(opcd=31, l=0, xog=0);

cmpi.set_asm("cmpwi %imm, %reg, %imm", bf, ra, si, l=0);
cmpi.set_asm("cmpi %reg, %imm, %reg, %imm", bf, l, ra, si);
cmpi.set_decoder(opcd=11, l=0);

cmpl.set_asm("cmplw %reg, %reg, %reg", bf, ra, rb, l=0);
```

```
cmpl.set_asm("cmpl %imm, %imm, %reg, %reg", bf, l, ra, rb);
cmpl.set_decoder(opcd=31, l=0, xog=32);

cmpli.set_asm("cmplwi %reg, %reg, %imm", bf, ra, ui, l=0);
cmpli.set_asm("cmpli %reg, %imm, %reg, %imm", bf, l, ra, ui);
cmpli.set_decoder(opcd=10, l=0);

cntlzw.set_asm("cntlzw %reg, %reg", ra, rs, rc=0);
cntlzw.set_asm("cntlzw. %reg, %reg", ra, rs, rc=1);
cntlzw.set_decoder(opcd=31, xog=26);

crand.set_asm("crand %reg, %reg, %reg", bt, ba, bb);
crand.set_decoder(opcd=19, xog=257);

crandc.set_asm("crandc %reg, %reg, %reg", bt, ba, bb);
crandc.set_decoder(opcd=19, xog=129);

creqv.set_asm("creqv %reg, %reg, %reg", bt, ba, bb);
creqv.set_decoder(opcd=19, xog=289);

crnand.set_asm("crnand %reg, %reg, %reg", bt, ba, bb);
crnand.set_decoder(opcd=19, xog=225);

crnor.set_asm("crnor %reg, %reg, %reg", bt, ba, bb);
crnor.set_decoder(opcd=19, xog=33);

cror.set_asm("cror %reg, %reg, %reg", bt, ba, bb);
cror.set_decoder(opcd=19, xog=449);

crorc.set_asm("crorc %reg, %reg, %reg", bt, ba, bb);
crorc.set_decoder(opcd=19, xog=417);

crxor.set_asm("crxor %reg, %reg, %reg", bt, ba, bb);
crxor.set_decoder(opcd=19, xog=193);
```

```
divw.set_asm("divw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
divw.set_asm("divw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
divwo.set_asm("divwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
divwo.set_asm("divwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
divw.set_decoder(opcd=31, xos=491);
```

```
divwu.set_asm("divwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
divwu.set_asm("divwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
divwuo.set_asm("divwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
divwuo.set_asm("divwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
divwu.set_decoder(opcd=31, xos=459);
```

```
eqv.set_asm("eqv %reg, %reg, %reg", ra, rs, rb, rc=0);
eqv.set_asm("eqv. %reg, %reg, %reg", ra, rs, rb, rc=1);
eqv.set_decoder(opcd=31, xog=284);
```

```
extsb.set_asm("extsb %reg, %reg", ra, rs, rc=0);
extsb.set_asm("extsb. %reg, %reg", ra, rs, rc=1);
extsb.set_decoder(opcd=31, xog=954);
```

```
extsh.set_asm("extsh %reg, %reg", ra, rs, rc=0);
extsh.set_asm("extsh. %reg, %reg", ra, rs, rc=1);
extsh.set_decoder(opcd=31, xog=922);
```

```
lbz.set_asm("lbz %reg, %imm(%reg)", rt, d, ra);
lbz.set_asm("lbz %reg, %exp@1(%reg)", rt, d, ra);
lbz.set_decoder(opcd=34);
```

```
lbzu.set_asm("lbzu %reg, %imm(%reg)", rt, d, ra);
lbzu.set_asm("lbzu %reg, %exp@1(%reg)", rt, d, ra);
lbzu.set_decoder(opcd=35);
```

```
lbzux.set_asm("lbzux %reg, %reg, %reg", rt, ra, rb);
lbzux.set_decoder(opcd=31, xog=119);
```



```
lbzx.set_asm("lbzx %reg, %reg, %reg", rt, ra, rb);
lbzx.set_decoder(opcd=31, xog=87);

lha.set_asm("lha %reg, %imm(%reg)", rt, d, ra);
lha.set_asm("lha %reg, %exp@l(%reg)", rt, d, ra);
lha.set_decoder(opcd=42);

lhau.set_asm("lhau %reg, %imm(%reg)", rt, d, ra);
lhau.set_asm("lhau %reg, %exp@l(%reg)", rt, d, ra);
lhau.set_decoder(opcd=43);

lhaux.set_asm("lhaux %reg, %reg, %reg", rt, ra, rb);
lhaux.set_decoder(opcd=31, xog=375);

lhax.set_asm("lhax %reg, %reg, %reg", rt, ra, rb);
lhax.set_decoder(opcd=31, xog=343);

lhbrx.set_asm("lhbrx %reg, %reg, %reg", rt, ra, rb);
lhbrx.set_decoder(opcd=31, xog=790);

lhz.set_asm("lhz %reg, %imm(%reg)", rt, d, ra);
lhz.set_asm("lhz %reg, %exp@l(%reg)", rt, d, ra);
lhz.set_decoder(opcd=40);

lhzu.set_asm("lhzu %reg, %imm(%reg)", rt, d, ra);
lhzu.set_asm("lhzu %reg, %exp@l(%reg)", rt, d, ra);
lhzu.set_decoder(opcd=41);

lhzux.set_asm("lhzux %reg, %reg, %reg", rt, ra, rb);
lhzux.set_decoder(opcd=31, xog=311);

lhzx.set_asm("lhzx %reg, %reg, %reg", rt, ra, rb);
lhzx.set_decoder(opcd=31, xog=279);

lmw.set_asm("lmw %reg, %imm(%reg)", rt, d, ra);
```

```
lmw.set_asm("lmw %reg, %exp@l(%reg)", rt, d, ra);
lmw.set_decoder(opcd=46);

lswi.set_asm("lswi %imm, %imm, %imm", rt, ra, nb);
lswi.set_decoder(opcd=31, xog=597);

lswx.set_asm("lswx %reg, %reg, %reg", rt, ra, rb);
lswx.set_decoder(opcd=31, xog=533);

lwbrx.set_asm("lwbrx %reg, %reg, %reg", rt, ra, rb);
lwbrx.set_decoder(opcd=31, xog=534);

lwz.set_asm("lwz %imm, %imm(%imm)", rt, d, ra);
lwz.set_asm("lwz %imm, %exp@l(%imm)", rt, d, ra);
lwz.set_decoder(opcd=32);

lwzu.set_asm("lwzu %reg, %imm(%reg)", rt, d, ra);
lwzu.set_asm("lwzu %reg, %exp@l(%reg)", rt, d, ra);
lwzu.set_decoder(opcd=33);

lwzux.set_asm("lwzux %reg, %reg, %reg", rt, ra, rb);
lwzux.set_decoder(opcd=31, xog=55);

lwzx.set_asm("lwzx %reg, %reg, %reg", rt, ra, rb);
lwzx.set_decoder(opcd=31, xog=23);

macchw.set_asm("macchw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchw.set_asm("macchw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
macchw.set_asm("macchw0 %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
macchw.set_asm("macchw0. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchw.set_decoder(opcd=4, xos=172);

macchws.set_asm("macchws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchws.set_asm("macchws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
macchws.set_asm("macchwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
```

```
macchws.set_asm("macchws. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchws.set_decoder(opcd=4, xos=236);
```

```
macchwsu.set_asm("macchwsu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchwsu.set_asm("macchwsu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
macchwsu.set_asm("macchwsuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
macchwsu.set_asm("macchwsuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchwsu.set_decoder(opcd=4, xos=204);
```

```
macchwu.set_asm("macchwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchwu.set_asm("macchwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
macchwu.set_asm("macchwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
macchwu.set_asm("macchwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchwu.set_decoder(opcd=4, xos=140);
```

```
machhw.set_asm("machhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
machhw.set_asm("machhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhw.set_asm("machhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhw.set_asm("machhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhw.set_decoder(opcd=4, xos=44);
```

```
machhws.set_asm("machhws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
machhws.set_asm("machhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhws.set_asm("machhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhws.set_asm("machhwsuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhws.set_asm("machhwsuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhws.set_decoder(opcd=4, xos=108);
```

```
machhwsu.set_asm("machhwsu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
machhwsu.set_asm("machhwsu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhwsu.set_asm("machhwsuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhwsu.set_asm("machhwsuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhwsu.set_decoder(opcd=4, xos=76);
```

```
machhwu.set_asm("machhwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
machhwu.set_asm("machhwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
```

```
machhwu.set_asm("machhwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhwu.set_asm("machhwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhwu.set_decoder(opcd=4, xos=12);
```

```
maclhw.set_asm("maclhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhw.set_asm("maclhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhw.set_asm("maclhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhw.set_asm("maclhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhw.set_decoder(opcd=4, xos=428);
```

```
maclhws.set_asm("maclhws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhws.set_asm("maclhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhws.set_asm("maclhwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhws.set_asm("maclhwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhws.set_decoder(opcd=4, xos=492);
```

```
maclhwsu.set_asm("maclhwsu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhwsu.set_asm("maclhwsu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhwsu.set_asm("maclhwsuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhwsu.set_asm("maclhwsuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhwsu.set_decoder(opcd=4, xos=460);
```

```
maclhwu.set_asm("maclhwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhwu.set_asm("maclhwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhwu.set_asm("maclhwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhwu.set_asm("maclhwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhwu.set_decoder(opcd=4, xos=396);
```

```
mcrf.set_asm("mcrf %imm, %imm", bf, bfa);
mcrf.set_decoder(opcd=19, xog=0);
```

```
mcrxr.set_asm("mcrxr %imm", bf);
mcrxr.set_decoder(opcd=31, xog=512);
```

```
mfcrr.set_asm("mfcrr %imm", rt);
```

```
mfcrr.set_decoder(opcd=31, xog=19);

/* INCOMPLETE IMPLEMENTATION! */
mfspr.set_asm("mfctr %imm", rt, sprf=0x120);
mfspr.set_asm("mflr %imm", rt, sprf=0x100);
mfspr.set_asm("mfspr %imm, %imm", rt, sprf);
mfspr.set_decoder(opcd=31, xog=339);

mtcrf.set_asm("mtcrf %imm, %imm", xfm, rs);
mtcrf.set_decoder(opcd=31, xog=144);

/* INCOMPLETE IMPLEMENTATION! */
mtspr.set_asm("mtctr %imm", rs, sprf=0x120);
mtspr.set_asm("mtlr %imm", rs, sprf=0x100);
mtspr.set_asm("mtspr %imm, %imm", sprf, rs);
mtspr.set_decoder(opcd=31, xog=467);

mulchw.set_asm("mulchw %reg, %reg, %reg", rt, ra, rb, rc=0);
mulchw.set_asm("mulchw. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulchw.set_decoder(opcd=4, xog=168);

mulchwu.set_asm("mulchwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mulchwu.set_asm("mulchwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulchwu.set_decoder(opcd=4, xog=136);

mulhhw.set_asm("mulhhw %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhhw.set_asm("mulhhw. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulhhw.set_decoder(opcd=4, xog=40);

mulhhwu.set_asm("mulhhwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhhwu.set_asm("mulhhwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulhhwu.set_decoder(opcd=4, xog=8);

mulhw.set_asm("mulhw %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhw.set_asm("mulhw. %reg, %reg, %reg", rt, ra, rb, rc=1);
```

```
mulhw.set_decoder(opcd=31, xos=75);

mulhwu.set_asm("mulhwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhwu.set_asm("mulhwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulhwu.set_decoder(opcd=31, xos=11);

mullhw.set_asm("mullhw %reg, %reg, %reg", rt, ra, rb, rc=0);
mullhw.set_asm("mullhw. %reg, %reg, %reg", rt, ra, rb, rc=1);
mullhw.set_decoder(opcd=4, xog=424);

mullhwu.set_asm("mullhwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mullhwu.set_asm("mullhwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mullhwu.set_decoder(opcd=4, xog=392);

mulli.set_asm("mulli %reg, %reg, %exp", rt, ra, d);
mulli.set_decoder(opcd=7);

mullw.set_asm("mullw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
mullw.set_asm("mullw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
mullw.set_asm("mullwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
mullw.set_asm("mullwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
mullw.set_decoder(opcd=31, xos=235);

nand.set_asm("nand %reg, %reg, %reg", ra, rs, rb, rc=0);
nand.set_asm("nand. %reg, %reg, %reg", ra, rs, rb, rc=1);
nand.set_decoder(opcd=31, xog=476);

neg.set_asm("neg %reg, %reg", rt, ra, oe=0, rc=0);
neg.set_asm("neg. %reg, %reg", rt, ra, oe=0, rc=1);
neg.set_asm("nego %reg, %reg", rt, ra, oe=1, rc=0);
neg.set_asm("nego. %reg, %reg", rt, ra, oe=1, rc=1);
neg.set_decoder(opcd=31, xos=104);

nmacchw.set_asm("nmacchw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmacchw.set_asm("nmacchw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
```

```
nmacchw.set_asm("nmacchwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmacchw.set_asm("nmacchwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmacchw.set_decoder(opcd=4, xos=174);

nmacchws.set_asm("nmacchws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmacchws.set_asm("nmacchws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmacchws.set_asm("nmacchwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmacchws.set_asm("nmacchwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmacchws.set_decoder(opcd=4, xos=238);

nmachhw.set_asm("nmachhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmachhw.set_asm("nmachhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmachhw.set_asm("nmachhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmachhw.set_asm("nmachhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmachhw.set_decoder(opcd=4, xos=46);

nmachhws.set_asm("nmachhws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmachhws.set_asm("nmachhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmachhws.set_asm("nmachhwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmachhws.set_asm("nmachhwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmachhws.set_decoder(opcd=4, xos=110);

nmaclhw.set_asm("nmaclhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmaclhw.set_asm("nmaclhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmaclhw.set_asm("nmaclhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmaclhw.set_asm("nmaclhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmaclhw.set_decoder(opcd=4, xos=430);

nmaclhws.set_asm("nmaclhws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmaclhws.set_asm("nmaclhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmaclhws.set_asm("nmaclhwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmaclhws.set_asm("nmaclhwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmaclhws.set_decoder(opcd=4, xos=494);

nor.set_asm("nor %reg, %reg, %reg", ra, rs, rb, rc=0);
```

```
nor.set_asm("nor. %reg, %reg, %reg", ra, rs, rb, rc=1);
nor.set_decoder(opcd=31, xog=124);

ore.set_asm("or %reg, %reg, %reg", ra, rs, rb, rc=0);
ore.set_asm("or. %reg, %reg, %reg", ra, rs, rb, rc=1);
ore.set_decoder(opcd=31, xog=444);

orc.set_asm("orc %reg, %reg, %reg", ra, rs, rb, rc=0);
orc.set_asm("orc. %reg, %reg, %reg", ra, rs, rb, rc=1);
orc.set_decoder(opcd=31, xog=412);

ori.set_asm("ori %reg, %reg, %imm", ra, rs, ui);
ori.set_decoder(opcd=24);

oris.set_asm("oris %reg, %reg, %imm", ra, rs, ui);
oris.set_decoder(opcd=25);

rlwimi.set_asm("rlwimi %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=0);
rlwimi.set_asm("rlwimi. %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=1);
rlwimi.set_decoder(opcd=20);

rlwinm.set_asm("rlwinm %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=0);
rlwinm.set_asm("rlwinm. %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=1);
rlwinm.set_decoder(opcd=21);

rlwnm.set_asm("rlwnm %imm, %imm, %imm, %imm, %imm", ra, rs, rb, mb,
me, rc=0);
rlwnm.set_asm("rlwnm. %imm, %imm, %imm, %imm, %imm", ra, rs, rb, mb,
me, rc=1);
rlwnm.set_decoder(opcd=23);
```



```
sc.set_asm("sc %imm", lev);
sc.set_decoder(opcd=17);

slw.set_asm("slw %reg, %reg, %reg", ra, rs, rb, rc=0);
slw.set_asm("slw. %reg, %reg, %reg", ra, rs, rb, rc=1);
slw.set_decoder(opcd=31, xog=24);

sraw.set_asm("sraw %reg, %reg, %reg", ra, rs, rb, rc=0);
sraw.set_asm("sraw. %reg, %reg, %reg", ra, rs, rb, rc=1);
sraw.set_decoder(opcd=31, xog=792);

srawi.set_asm("srawi %reg, %reg, %reg", ra, rs, sh, rc=0);
srawi.set_asm("srawi. %reg, %reg, %reg", ra, rs, sh, rc=1);
srawi.set_decoder(opcd=31, xog=824);

srw.set_asm("srw %reg, %reg, %reg", ra, rs, rb, rc=0);
srw.set_asm("srw. %reg, %reg, %reg", ra, rs, rb, rc=1);
srw.set_decoder(opcd=31, xog=536);

stb.set_asm("stb %reg, %imm(%reg)", rs, d, ra);
stb.set_asm("stb %reg, %exp@l(%reg)", rs, d, ra);
stb.set_decoder(opcd=38);

stbu.set_asm("stbu %reg, %imm(%reg)", rs, d, ra);
stbu.set_asm("stbu %reg, %exp@l(%reg)", rs, d, ra);
stbu.set_decoder(opcd=39);

stbux.set_asm("stbux %reg, %reg, %reg", rs, ra, rb);
stbux.set_decoder(opcd=31, xog=247);

stbx.set_asm("stbx %reg, %reg, %reg", rs, ra, rb);
stbx.set_decoder(opcd=31, xog=215);

sth.set_asm("sth %reg, %imm(%reg)", rs, d, ra);
sth.set_asm("sth %reg, %exp@l(%reg)", rs, d, ra);
```

```
sth.set_decoder(opcd=44);

sthbrx.set_asm("sthbrx %reg, %reg, %reg", rs, ra, rb);
sthbrx.set_decoder(opcd=31, xog=918);

sthu.set_asm("sthu %reg, %imm(%reg)", rs, d, ra);
sthu.set_asm("sthu %reg, %exp@l(%reg)", rs, d, ra);
sthu.set_decoder(opcd=45);

sthux.set_asm("sthux %reg, %reg, %reg", rs, ra, rb);
sthux.set_decoder(opcd=31, xog=439);

sthx.set_asm("sthx %reg, %reg, %reg", rs, ra, rb);
sthx.set_decoder(opcd=31, xog=407);

stmw.set_asm("stmw %reg, %imm(%reg)", rs, d, ra);
stmw.set_asm("stmw %reg, %exp@l(%reg)", rs, d, ra);
stmw.set_decoder(opcd=47);

stswi.set_asm("stswi %reg, %reg, %reg", rs, ra, nb);
stswi.set_decoder(opcd=31, xog=725);

stswx.set_asm("stswx %reg, %reg, %reg", rs, ra, rb);
stswx.set_decoder(opcd=31, xog=661);

stw.set_asm("stw %reg, %imm(%reg)", rs, d, ra);
stw.set_asm("stw %reg, %exp@l(%reg)", rs, d, ra);
stw.set_decoder(opcd=36);

stwbrx.set_asm("stwbrx %reg, %reg, %reg", rs, ra, rb);
stwbrx.set_decoder(opcd=31, xog=662);

stwu.set_asm("stwu %reg, %imm(%reg)", rs, d, ra);
stwu.set_asm("stwu %reg, %imm(%reg)", rs, d, ra);
stwu.set_decoder(opcd=37);
```

```
stwux.set_asm("stwux %reg, %reg, %reg", rs, ra, rb);
stwux.set_decoder(opcd=31, xog=183);

stwx.set_asm("stwx %reg, %reg, %reg", rs, ra, rb);
stwx.set_decoder(opcd=31, xog=151);

subf.set_asm("subf %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
subf.set_asm("subf. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
subf.set_asm("subfo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
subf.set_asm("subfo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
subf.set_decoder(opcd=31, xos=40);

subfc.set_asm("subfc %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
subfc.set_asm("subfc. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
subfc.set_asm("subfco %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
subfc.set_asm("subfco. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
subfc.set_decoder(opcd=31, xos=8);

subfe.set_asm("subfe %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
subfe.set_asm("subfe. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
subfe.set_asm("subfeo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
subfe.set_asm("subfeo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
subfe.set_decoder(opcd=31, xos=136);

subfic.set_asm("subfic %reg, %reg, %exp", rt, ra, d);
subfic.set_decoder(opcd=8);

subfme.set_asm("subfme %reg, %reg", rt, ra, oe=0, rc=0);
subfme.set_asm("subfme. %reg, %reg", rt, ra, oe=0, rc=1);
subfme.set_asm("subfmeo %reg, %reg", rt, ra, oe=1, rc=0);
subfme.set_asm("subfmeo. %reg, %reg", rt, ra, oe=1, rc=1);
subfme.set_decoder(opcd=31, xos=232);

subfze.set_asm("subfze %reg, %reg", rt, ra, oe=0, rc=0);
```

```

subfze.set_asm("subfze. %reg, %reg", rt, ra, oe=0, rc=1);
subfze.set_asm("subfzeo %reg, %reg", rt, ra, oe=1, rc=0);
subfze.set_asm("subfzeo. %reg, %reg", rt, ra, oe=1, rc=1);
subfze.set_decoder(opcd=31, xos=200);

```

```

xxor.set_asm("xor %reg, %reg, %reg", ra, rs, rb, rc=0);
xxor.set_asm("xor. %reg, %reg, %reg", ra, rs, rb, rc=1);
xxor.set_decoder(opcd=31, xog=316);

```

```

xori.set_asm("xori %reg, %reg, %imm", ra, rs, ui);
xori.set_decoder(opcd=26);

```

```

xoris.set_asm("xoris %reg, %reg, %imm", ra, rs, ui);
xoris.set_decoder(opcd=27);

```

```

/*****/
/* Synthetic Instructions      */
/*****/

```

```

pseudo_instr("mr %reg, %reg") {
    "or %0, %1, %1";
}

```

```

pseudo_instr("subi %reg, %reg, %imm") {
    "addi %0, %1, -%2";
}

```

```

pseudo_instr("subic %reg, %reg, %imm") {
    "addic %0, %1, -%2";
}

```

```

pseudo_instr("subic. %reg, %reg, %imm") {
    "addic. %0, %1, -%2";
}

```

```
}

pseudo_instr("subis %reg, %reg, %imm") {
    "addis %0, %1, -%2";
}

pseudo_instr("srwi %reg, %reg, %imm") {
    "rlwinm %0, %1, 32-%2, %2, 31";
}

pseudo_instr("crnot %reg, %reg") {
    "crnor %0, %1, %1";
}

pseudo_instr("insrwi %reg, %reg, %imm, %imm") {
    "rlwimi %0, %1, 32-%3-%2, %3, %3+%2-1";
}

pseudo_instr("slwi %reg, %reg, %imm") {
    "rlwinm %0, %1, %2, 0, 31-%2";
}

pseudo_instr("ble %imm, %addr") {
    "bc 0x4, %0*4+1, %1";
}

pseudo_instr("bne %imm, %addr") {
    "bc 0x4, %0*4+2, %1";
}

pseudo_instr("bgt %imm, %addr") {
    "bc 0xC, %0*4+1, %1";
}

pseudo_instr("blt %imm, %addr") {
```

```

        "bc 0xC, %0*4, %1";
    }

    pseudo_instr("bge %imm, %addr") {
        "bc 0x4, %0*4, %1";
    }

    pseudo_instr("beq %imm, %addr") {
        "bc 0xC, %0*4+2, %1";
    }

};

};

```

5.1.3 ppc405_syscall.cpp

```

/*****/
/* The ArchC PowerPC405 functional model. */
/* System Call Functions Implementation */
/* Author: Bruno Corsi dos Santos */
/*      Alexandre Baldassin (assembly information) */
/*      Sandro Carvalho (PowerPC405 instructions) */
/* */
/* For more information on ArchC, please visit: */
/* http://www.archc.org */
/* */
/* The ArchC Team */
/* Computer Systems Laboratory (LSC) */
/* IC-UNICAMP */
/* http://www.lsc.ic.unicamp.br */
/* */
/* System Design Automation Lab (LAPS) */
/* INF-UFSC */
/* http://www.laps.inf.ufsc.br */

```

```
/******  
  
#include "ppc405_syscall.H"  
#include "ac_resources.H"  
  
void ppc405_syscall::get_buffer(int argn, unsigned char* buf, unsigned int size)  
{  
    unsigned int addr = GPR.read(3+argn);  
  
    for (unsigned int i = 0; i<size; i++, addr++) {  
        buf[i] = MEM.read_byte(addr);  
    }  
}  
  
void ppc405_syscall::set_buffer(int argn, unsigned char* buf, unsigned int size)  
{  
    unsigned int addr = GPR.read(3+argn);  
  
    for (unsigned int i = 0; i<size; i++, addr++) {  
        MEM.write_byte(addr, buf[i]);  
    }  
}  
  
void ppc405_syscall::set_buffer_noinvert(int argn, unsigned char* buf, unsigned  
int size)  
{  
    unsigned int addr = GPR.read(3+argn);  
  
    for (unsigned int i = 0; i<size; i+=4, addr+=4) {  
        MEM.write(addr, *(unsigned int *) &buf[i]);  
    }  
}  
  
int ppc405_syscall::get_int(int argn)
```

```
{
    return GPR.read(3+argn);
}

void ppc405_syscall::set_int(int argn, int val)
{
    GPR.write(3+argn, val);
}

void ppc405_syscall::return_from_syscall()
{
    unsigned int oldr1;
    unsigned int oldr31;
    oldr1=MEM.read(GPR.read(1));
    oldr31=MEM.read(GPR.read(1)+28);
    GPR.write(1,oldr1);
    GPR.write(31,oldr31);
    ac_resources::ac_pc=LR.read();
}

void ppc405_syscall::set_prog_args(int argc, char **argv)
{
    extern unsigned AC_RAM_END;
    int i, j, base;

    unsigned int ac_argv[30];
    char ac_argstr[512];

    base = AC_RAM_END - 512;
    for (i=0, j=0; i<argc; i++) {
        int len = strlen(argv[i]) + 1;
        ac_argv[i] = base + j;
        memcpy(&ac_argstr[j], argv[i], len);
        j += len;
    }
}
```



```
//Write argument string
GPR.write(3, AC_RAM_END-512);
set_buffer(0, (unsigned char*) ac_argstr, 512);

//Write string pointers
GPR.write(3, AC_RAM_END-512-120);
set_buffer_noinvert(0, (unsigned char*) ac_argv, 120);

//Set r3 to the argument count
GPR.write(3, argc);

//Set r4 to the string pointers
GPR.write(4, AC_RAM_END-512-120);
}
```

5.1.4 ppc405_gdb_funcs.cpp

```
/*
*****
*/
/* The ArchC PowerPC405 functional model. */
/* Gnu Debugger Functions Implementation */
/* Author: Bruno Corsi dos Santos */
/*      Alexandre Baldassin (assembly information) */
/*      Sandro Carvalho (PowerPC405 instructions) */
/* */
/* For more information on ArchC, please visit: */
/* http://www.archc.org */
/* */
/* The ArchC Team */
/* Computer Systems Laboratory (LSC) */
/* IC-UNICAMP */
/* http://www.lsc.ic.unicamp.br */
/* */
/* System Design Automation Lab (LAPS) */
/* INF-UFSC */
*/
```

```
/* http://www.laps.inf.ufsc.br */
/*****/

#include "ppc405.H"

int ppc405::nRegs(void) {
    return 104;
}

ac_word ppc405::reg_read( int reg ) {
    unsigned int n;

    if ( ( reg >= 0 ) && ( reg < 32 ) )
        return GPR.read( reg );
    else {
        switch (reg) {

            case 96:
                n=ac_resources::ac_pc;
                break;

            /* case ??:
                n=MSR.read();
                break; */

            case 98:
                n=CR.read();
                break;

            case 99:
                n=LR.read();
                break;
```

```
        case 100:
            n=CTR.read();
            break;

        case 101:
            n=XER.read();
            break;

        default:
            return 0;
            break;

    }

    return(n);

}

}

void ppc405::reg_write( int reg, ac_word value ) {

    /* general purpose registers */
    if ( ( reg >= 0 ) && ( reg < 32 ) )
        GPR.write(reg,value);
    else {
        switch (reg) {

            case 96:
                ac_resources::ac_pc=value;
                break;

            /*case ??:
                MSR.write(value);
                break;

        }
    }
}
```

```
    */

    case 98:
        CR.write(value);
        break;

    case 99:
        LR.write(value);
        break;

    case 100:
        CTR.write(value);
        break;

    case 101:
        XER.write(value);
        break;

    default:
        /* No completely implemented register */
        break;
}
}
}

unsigned char ppc405::mem_read( unsigned int address ) {
    return ac_resources::IM->read_byte( address );
}

void ppc405::mem_write( unsigned int address, unsigned char byte ) {
    ac_resources::IM->write_byte( address, byte );
}
```

5.1.5 ppc405-isa.cpp

```
/* *****  
/* The ArchC PowerPC405 functional model. */  
/* Instruction Set Architecture Implementation */  
/* Author: Bruno Corsi dos Santos */  
/*      Alexandro Baldassin (assembly information) */  
/*      Sandro Carvalho (PowerPC405 instructions) */  
/* */  
/* For more information on ArchC, please visit: */  
/* http://www.archc.org */  
/* */  
/* The ArchC Team */  
/* Computer Systems Laboratory (LSC) */  
/* IC-UNICAMP */  
/* http://www.lsc.ic.unicamp.br */  
/* */  
/* System Design Automation Lab (LAPS) */  
/* INF-UFSC */  
/* http://www.laps.inf.ufsc.br */  
/* *****  
  
//IMPLEMENTATION NOTES:  
// PowerPC 32 bits family.  
// The PowerPC 405 instruction family are implemented.  
// Based on IBM and Xilinx manuals of PowerPC 405.  
// mtspr and mfspr instructions not completely implemented.  
// sc instruction not completely implemented and never used.  
  
#include "ppc405-isa.H"  
#include "ac_isa_init.cpp"  
  
//If you want debug information for this model, uncomment next line  
//#define DEBUG_MODEL
```

```
#include "ac_debug_model.H"

//Compute CRO fields LT, GT, EQ, SO
//XER.SO must be updated by instruction before the use of this routine!
//Arguments:
//int result -> The result register
inline void CR0_update(unsigned int result) {

    /* LT field */
    if((result & 0x80000000) >> 31)
        CR.write(CR.read() | 0x80000000); /* 1 */
    else
        CR.write(CR.read() & 0x7FFFFFFF); /* 0 */

    /* GT field */
    if(((~result & 0x80000000) >> 31) && (result!=0))
        CR.write(CR.read() | 0x40000000); /* 1 */
    else
        CR.write(CR.read() & 0xBFFFFFFF); /* 0 */

    /* EQ field */
    if(result==0)
        CR.write(CR.read() | 0x20000000); /* 1 */
    else
        CR.write(CR.read() & 0xDFFFFFFF); /* 0 */

    /* SO field */
    if(XER.read() & 0x80000000)
        CR.write(CR.read() | 0x10000000); /* 1 */
    else
        CR.write(CR.read() & 0xEFFFFFFF); /* 0 */
}
```

```
//Compute XER overflow fields SO, OV
//Arguments:
//int result -> The result register
//int s1 -> Source 1
//int s2 -> Source 2
//int s3 -> Source 3 (if only two sources, use 0)
inline void add_XER_OV_SO_update(int result,int s1,int s2,int s3) {

    long long int longresult =
        (long long int)(int)s1 + (long long int)(int)s2 + (long long int)(int)s3;

    if(longresult != (long long int)(int)result) {
        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    }
    else {
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    }
}

//Compute XER carry field CA
//Arguments:
//int result -> The result register
//int s1 -> Source 1
//int s2 -> Source 2
//int s3 -> Source 3 (if only two sources, use 0)
inline void add_XER_CA_update(int result,int s1,int s2,int s3) {

    unsigned long long int longresult =
        (unsigned long long int)(unsigned int)s1 +
        (unsigned long long int)(unsigned int)s2 +
        (unsigned long long int)(unsigned int)s3;

    if(longresult > 0xFFFFFFFF)
```

```

        XER.write(XER.read() |0x20000000); /* Write 1 to bit 2 CA */
    else
        XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */

}

//Compute XER overflow fields SO, OV
//Arguments:
//int result -> The result register
//int s1 -> Source 1
//int s2 -> Source 2
inline void divws_XER_OV_SO_update(int result,int s1,int s2) {

    long long int longresult =
        (long long int)(int)s1 / (long long int)(int)s2;

    if(longresult != (long long int)(int)result) {
        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    }
    else
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */

}

//Compute XER overflow fields SO, OV
//Arguments:
//int result -> The result register
//int s1 -> Source 1
//int s2 -> Source 2
inline void divwu_XER_OV_SO_update(int result,int s1,int s2) {

    unsigned long long int longresult =
        (unsigned long long int)(unsigned int)s1 /

```



```

        (unsigned long long int)(unsigned int)s2;

    if(longresult != (unsigned long long int)(unsigned int)result) {
        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    }
    else
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
}

//Ceil function
inline int ceil(int value, int divisor) {
    int res;
    if ((value % divisor)!=0)
        res=int(value/divisor)+1;
    else res=value/divisor;
    return res;
}

//Rotl function
inline unsigned int rotl(unsigned int reg,unsigned int n) {
    unsigned int tmp1=reg;
    unsigned int tmp2=reg;
    unsigned int rotated=(tmp1 << n) |(tmp2 >> 32-n);

    return(rotated);
}

//Mask32rlw function
inline unsigned int mask32rlw(unsigned int i,unsigned int f) {

    unsigned int maski,maskf;

    if(i<=f) {

```

```

    maski=(0xFFFFFFFF>>i);
    maskf=(0xFFFFFFFF<<(31-f));
    return(maski & maskf);
}
else {
    maski=(0xFFFFFFFF>>f);
    maski=maski>>1;
    maskf=(0xFFFFFFFF<<(31-i));
    maskf=maskf<<1;
    return(~(maski & maskf));
}
}

//Function dump General Purpose Registers
inline void dumpGPR() {
    int i;
    for(i=0 ; i<32 ; i++)
        dbg_printf("r%d -> %#x \n",i,GPR[i]);
}

//Function that returns the value of XER TBC
inline unsigned int XER_TBC_read() {
    return(XER.read() & 0x0000007F);
}

//Function that returns the value of XER OV
inline unsigned int XER_OV_read() {
    if(XER.read() & 0x40000000)
        return 1;
    else
        return 0;
}

```

```
//Function that returns the value of XER SO
inline unsigned int XER_SO_read() {
    if(XER.read() & 0x80000000)
        return 1;
    else
        return 0;
}

//Function that returns the value of XER CA
inline unsigned int XER_CA_read() {
    if(XER.read() & 0x20000000)
        return 1;
    else
        return 0;
}

//Function dump various registers
inline void dumpREG() {
    dbg_printf("XER.OV = %d\n",XER_OV_read());
    dbg_printf("XER.SO = %d\n",XER_SO_read());
    dbg_printf("XER.CA = %d\n",XER_CA_read());
    dbg_printf("CR = %#x\n",CR.read());
    dbg_printf("LR = %#x\n",LR.read());
    dbg_printf("CTR = %#x\n",CTR.read());
}

enum macType { NORMAL , NEGATIVE , MULTIPLY };

enum macOrder { CROSS , HIGH , LOW };

enum macOverflow { MODULE , SATURATE };
```

```

enum macSign { SIGNED , UNSIGNED };

void genericMac ( macType    tp ,
                 macOrder   od ,
                 macOverflow of ,
                 macSign     sn ,
                 unsigned    oe ,
                 unsigned    rc ,
                 int         rt ,
                 int         ra ,
                 int         rb )
{
    ac_UHword value1    = 0; // first halfword operand
    ac_UHword value2    = 0; // second halfword operand
    ac_Uword  result    = 0; // result word
    ac_Uword  prod      = 0;
    ac_UDword longResult = 0;

    switch ( od )
    {
        case CROSS:
            value1 = (ac_UHword)(GPR[ra] & 0x0000FFFF); // low-order halfword
            value2 = (ac_UHword)(GPR[rb] >> 16);      // high-order halfword
            break;
        case HIGH:
            value1 = (ac_UHword)(GPR[ra] >> 16);      // high-order halfword
            value2 = (ac_UHword)(GPR[rb] >> 16);      // high-order halfword
            break;
        case LOW:
            value1 = (ac_UHword)(GPR[ra] & 0x0000FFFF); // low-order halfword
            value2 = (ac_UHword)(GPR[rb] & 0x0000FFFF); // low-order halfword
            break;
    }

    switch ( sn )

```

```

{
    case SIGNED:
        prod = (ac_Sword)((ac_SHword)value1) * (ac_Sword)((ac_SHword)value2);
        break;
    case UNSIGNED:
        prod = (ac_Uword)value1 * (ac_Uword)value2;
        break;
}

switch ( tp )
{
    case NORMAL:
        longResult = (ac_SDword)((ac_Sword)GPR[rt]) + (ac_SDword)((ac_Sword)prod);
        break;
    case NEGATIVE:
        longResult = (ac_SDword)((ac_Sword)GPR[rt]) - (ac_SDword)((ac_Sword)prod);
        break;
    case MULTIPLY:
        longResult = prod;
        break;
}

if ( of == SATURATE)
{
    ac_SDword smaxValue = 0x80000000; // 2^31

    ac_UDword umaxValue = 0xFFFFFFFF; // 2^32-1

    switch ( sn )
    {
        case SIGNED:
            if ((ac_SDword)longResult < -smaxValue )
                {printf("longResult < -smaxValue\n");
                longResult = -smaxValue;}
            if ((ac_SDword)longResult > smaxValue-1 )

```

```
        {printf("longResult > smaxValue-1\n");
         longResult = smaxValue-1;}
        break;
    case UNSIGNED:
        if ( longResult > umaxValue )
            {printf("longResult > umaxValue\n");
             longResult = umaxValue;}
        break;
    }
}

result = longResult;

GPR[rt] = result;

if ( oe == 1 )
{
    if(longResult != (ac_UDword)result)
    {

        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    } else {
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    }
}

if ( rc == 1 )
{
    CR0_update(result);
}
}
```

```
//!Generic instruction behavior method.
void ac_behavior( instruction )
{
    dbg_printf("\n program counter=%#x\n", (int)ac_pc);
    ac_pc+=4;
    //dumpGPR();
    //dumpREG();
}

//!Generic begin behavior method.
void ac_behavior( begin )
{
    dbg_printf("Starting simulator...\n");

    /* Here the stack is started in a */
    GPR[1] = AC_RAM_END - 1024;

    /* Make a jump out of memory if it doesn't have an abi */
    LR.write(0xFFFFFFFF);
}

//! Instruction Format behavior methods.
void ac_behavior( I1 ){}
void ac_behavior( B1 ){}
void ac_behavior( SC1 ){}
void ac_behavior( D1 ){}
void ac_behavior( D2 ){}
void ac_behavior( D3 ){}
void ac_behavior( D4 ){}
void ac_behavior( D5 ){}
void ac_behavior( D6 ){}
void ac_behavior( D7 ){}
void ac_behavior( X1 ){}

```

```
void ac_behavior( X2 ){}
void ac_behavior( X3 ){}
void ac_behavior( X4 ){}
void ac_behavior( X5 ){}
void ac_behavior( X6 ){}
void ac_behavior( X7 ){}
void ac_behavior( X8 ){}
void ac_behavior( X9 ){}
void ac_behavior( X10 ){}
void ac_behavior( X11 ){}
void ac_behavior( X12 ){}
void ac_behavior( X13 ){}
void ac_behavior( X14 ){}
void ac_behavior( X15 ){}
void ac_behavior( X16 ){}
void ac_behavior( X17 ){}
void ac_behavior( X18 ){}
void ac_behavior( X19 ){}
void ac_behavior( X20 ){}
void ac_behavior( X21 ){}
void ac_behavior( X22 ){}
void ac_behavior( X23 ){}
void ac_behavior( X24 ){}
void ac_behavior( X25 ){}
void ac_behavior( XL1 ){}
void ac_behavior( XL2 ){}
void ac_behavior( XL3 ){}
void ac_behavior( XL4 ){}
void ac_behavior( XFX1 ){}
void ac_behavior( XFX2 ){}
void ac_behavior( XFX3 ){}
void ac_behavior( XFX4 ){}
void ac_behavior( XFX5 ){}
void ac_behavior( X01 ){}
void ac_behavior( X02 ){}

```



```

void ac_behavior( X03 ){}
void ac_behavior( M1 ){}
void ac_behavior( M2 ){}

//!Instruction add behavior method.
// Add
void ac_behavior( add )
{
    dbg_printf(" add%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt

    int result = GPR[ra] + GPR[rb];

    if (oe==1)
        add_XER_OV_SO_update(result, GPR[ra], GPR[rb], 0);

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction addc behavior method.
// Add Carrying
void ac_behavior( addc )
{
    dbg_printf(" addc%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt

    int result = GPR[ra] + GPR[rb];

    add_XER_CA_update(result, GPR[ra], GPR[rb], 0);

    if (oe==1)
        add_XER_OV_SO_update(result, GPR[ra], GPR[rb], 0);

```

```

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction adde behavior method.
// Add Extended
void ac_behavior( adde )
{
    dbg_printf(" adde%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r,
    rt, ra, rb);

    int result = GPR[ra] + GPR[rb] + XER_CA_read();

    add_XER_CA_update(GPR[rt], GPR[ra], GPR[rb], XER_CA_read());

    if (oe==1)
        add_XER_OV_SO_update(result, GPR[ra], GPR[rb], XER_CA_read());
    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction addi behavior method.
// Add Immediate
void ac_behavior( addi )
{
    dbg_printf(" addi r%d, r%d, %d\n\n", rt, ra, d);

    int ime32 = d;

    if (ra == 0)
        GPR[rt] = ime32;
    else

```

```
        GPR[rt] = GPR[ra] + ime32;

};

//!Instruction addic behavior method.
// Add Immediate Carrying
void ac_behavior( addic )
{
    dbg_printf(" addic r%d, r%d, %d\n\n",rt,ra,d);

    int ime32 = d;

    int result = GPR[ra] + ime32;

    add_XER_CA_update(result,GPR[ra],ime32,0);

    GPR[rt] = result;
};

//!Instruction addic_behavior method.
// Add Immediate Carrying and Record
void ac_behavior( addic_)
{
    dbg_printf(" addic.  r%d, r%d, %d\n\n",rt,ra,d);

    int ime32 = d;

    int result = GPR[ra] + ime32;

    add_XER_CA_update(result,GPR[ra],ime32,0);

    CR0_update(result); // Do not have rc field and update CR0

    GPR[rt] = result;
};
```

```
//!Instruction addis behavior method.
// Add Immediate Shifted
void ac_behavior( addis )
{
    dbg_printf(" addis r%d, r%d, %d\n\n",rt,ra,d);

    int ime32 = d;

    ime32 = ime32 << 16;

    if (ra == 0)
        GPR[rt] = ime32;
    else
        GPR[rt] = GPR[ra] + ime32;
};

//!Instruction addme behavior method.
// Add to Minus One Extended
void ac_behavior( addme )
{
    dbg_printf(" addme%s r%d, r%d\n\n",(oe==1?"o.":"o"):(rc==1?".":"")),rt,ra);

    int result = GPR[ra] + XER_CA_read() + (-1);

    add_XER_CA_update(result,GPR[ra],XER_CA_read(),-1);

    if (oe==1)
        add_XER_OV_SO_update(result,GPR[ra],XER_CA_read(),-1);

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};
```

```

//!Instruction addze behavior method.
// Add to Zero Extended
void ac_behavior( addze )
{
    dbg_printf(" addze%s r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt,ra);

    int result = GPR[ra] + XER_CA_read();

    add_XER_CA_update(result,GPR[ra],XER_CA_read(),0);

    if (oe==1)
        add_XER_OV_SO_update(result,GPR[ra],XER_CA_read(),0);

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction ande behavior method.
// AND
void ac_behavior( ande )
{
    dbg_printf(" and%s r%d, r%d, r%d\n\n", (rc==1?".":""),ra,rs,rb);

    int result = GPR[rs] & GPR[rb];

    if (rc==1)
        CR0_update(result);

    GPR[ra] = result;
};

//!Instruction andc behavior method.

```

```
// AND with Complement
void ac_behavior( andc )
{
    dbg_printf(" andc%s r%d, r%d, r%d\n\n", (rc==1?".":""), ra, rs, rb);

    int result = GPR[rs] & ~GPR[rb];

    if (rc==1)
        CR0_update(result);

    GPR[ra] = result;
};

//!Instruction andi_behavior method.
// AND Immediate
void ac_behavior( andi_)
{
    dbg_printf(" andi.  r%d, r%d, %d\n\n", ra, rs, ui);

    unsigned int ime32 = (unsigned short int)ui;

    int result = GPR[rs] & ime32;

    CR0_update(result); // Do not have rc field and update CR0

    GPR[ra] = result;
};

//!Instruction andis_behavior method.
// AND Immediate Shifted
void ac_behavior( andis_)
{
    dbg_printf(" andis.  r%d, r%d, %d\n\n", ra, rs, ui);

    unsigned int ime32=(unsigned short int)ui;
```

```
ime32 = ime32 << 16;

int result = GPR[rs] & ime32;

CR0_update(result); // Do not have rc field and update CR0

GPR[ra] = result;
};

//!Instruction b behavior method.
// Branch
void ac_behavior( b )
{
    dbg_printf(" b%s %d\n\n", (lk==1?(aa==1?"la":"l"):(aa==1?"a":"")), li);

    int displacement;
    unsigned int nia;

    ac_pc-=4; /* Because pre-increment */

    displacement = li << 2;

    nia = displacement;

    if(aa==0)
        nia += ac_pc;

    if(lk==1)
        LR.write(ac_pc + 4);

    ac_pc = nia;
};
```

```

//!Instruction bc behavior method.
// Branch Conditional
void ac_behavior( bc )
{
    dbg_printf(" bc%s %d, %d, %d\n\n", (lk==1?(aa==1?"la":"l"):(aa==1?"a":"")), bo, bi, l

    int displacement;
    unsigned int nia;
    unsigned int masc;

    masc = 0x80000000 >> bi;

    ac_pc -= 4; /* Because pre-increment */

    if ((bo & 0x04) == 0x00)
        CTR.write(CTR.read()-1);

    if (((bo & 0x04) || /* Branch */
        ((CTR.read()==0) && (bo & 0x02)) ||
        (!(CTR.read()==0) && !(bo & 0x02)))
        &&
        ((bo & 0x10) ||
        (((CR.read() & masc) && (bo & 0x08)) ||
        (!(CR.read() & masc) && !(bo & 0x08)))))) {

        displacement = bd<<2;

        nia = displacement;

        if (aa == 0)
            nia += ac_pc;
    }
    else { /* No branch */
        nia = ac_pc + 4;
    }
}

```



```

    if (lk==1)
        LR.write(ac_pc + 4);

    ac_pc = nia;
};

//!Instruction bcctr behavior method.
// Branch Conditional to Count Register
void ac_behavior( bcctr )
{
    dbg_printf(" bcctr%s %d, %d\n\n", (lk==1?"1":""), bo, bi);

    unsigned int nia;
    unsigned int masc;

    masc = 0x80000000 >> bi;

    ac_pc -= 4; /* Because pre-increment */

    if ((bo & 0x04) == 0x00)
        CTR.write(CTR.read()-1);

    if (((bo & 0x04) || /* Branch */
        ((CTR.read()==0) && (bo & 0x02)) ||
        (!(CTR.read()==0) && !(bo & 0x02)))
        &&
        ((bo & 0x10) ||
        (((CR.read() & masc) && (bo & 0x08)) ||
        (!(CR.read() & masc) && !(bo & 0x08)))))) {

        nia = CTR.read() & 0xFFFFFFFF;

    }
    else { /* No Branch */

```

```

        nia = ac_pc + 4;
    }

    if(lk==1)
        LR.write(ac_pc + 4);

    ac_pc = nia;
};

//!Instruction bclr behavior method.
// Branch Conditional to Link Register
void ac_behavior( bclr )
{
    dbg_printf(" bclr%s %d, %d\n\n", (lk==1?"1":""), bo, bi);

    unsigned int nia;
    unsigned int masc;

    masc = 0x80000000 >> bi;

    ac_pc -= 4; /* Because pre-increment */

    if((bo & 0x04) == 0x00)
        CTR.write(CTR.read()-1);

    if(((bo & 0x04) || /* Branch */
        ((CTR.read()==0) && (bo & 0x02)) ||
        (!(CTR.read()==0) && !(bo & 0x02)))
        &&
        ((bo & 0x10) ||
        (((CR.read() & masc) && (bo & 0x08)) ||
        (!(CR.read() & masc) && !(bo & 0x08)))))) {

        nia = LR.read() & 0xFFFFFFFF;
    }
}

```

```
    }
    else { /* No Branch */
        nia = ac_pc + 4;
    }

    if(lk==1)
        LR.write(ac_pc+4);

    ac_pc = nia;
};

//!Instruction cmp behavior method.
// Compare
void ac_behavior( cmp )
{
    dbg_printf(" cmp crf%d, 0, r%d, r%d\n\n",bf,ra,rb);

    unsigned int c=0x00;
    unsigned int n=bf;
    unsigned int masc=0xF0000000;

    if((int)GPR[ra] < (int)GPR[rb])
        c = c |0x80000000;
    if((int)GPR[ra] > (int)GPR[rb])
        c = c |0x40000000;
    if((int)GPR[ra] == (int)GPR[rb])
        c = c |0x20000000;
    if(XER_SO_read()==1)
        c = c |0x10000000;

    c = c >> (n*4);
    masc = ~(masc >> (n*4));

    CR.write((CR.read() & masc) |c);
```

```
};

//!Instruction cmpi behavior method.
// Compare Immediate
void ac_behavior( cmpi )
{
    dbg_printf(" cmpi crf%d, 0, r%d, %d\n\n",bf,ra,si);

    unsigned int c=0x00;
    unsigned int n=bf;
    unsigned int masc=0xF0000000;

    int ime32=(short int)(si);

    if((int)GPR[ra] < ime32)
        c = c |0x80000000;
    if((int)GPR[ra] > ime32)
        c = c |0x40000000;
    if((int)GPR[ra] == ime32)
        c = c |0x20000000;
    if(XER_SO_read()==1)
        c = c |0x10000000;

    c = c >> (n*4);
    masc = ~(masc >> (n*4));

    CR.write((CR.read() & masc) |c);
};

//!Instruction cmpl behavior method.
// Compare Logical
void ac_behavior( cmpl )
{
    dbg_printf(" cmpl crf%d, 0, r%d, r%d\n\n",bf,ra,rb);
```

```

unsigned int c=0x00;
unsigned int n=bf;
unsigned int masc=0xF0000000;

unsigned int uintra=GPR[ra];
unsigned int uintrb=GPR[rb];

if(uintra < uintrb)
    c = c |0x80000000;
if(uintra > uintrb)
    c = c |0x40000000;
if(uintra == uintrb)
    c = c |0x20000000;
if(XER_SO_read()==1)
    c = c |0x10000000;

c = c >> (n*4);
masc = ~(masc >> (n*4));

CR.write((CR.read() & masc) |c);

};

//!Instruction cmpli behavior method.
// Compare Logical Immediate
void ac_behavior( cmpli )
{
    dbg_printf(" cmpli crf%d, 0, r%d, %d\n\n",bf,ra,ui);

    unsigned int c=0x00;
    unsigned int n=bf;
    unsigned int masc=0xF0000000;

    unsigned int ime32=(unsigned short int)(ui);

```

```

if(GPR[ra] < ime32)
    c = c |0x80000000;
if(GPR[ra] > ime32)
    c = c |0x40000000;
if(GPR[ra] == ime32)
    c = c |0x20000000;
if(XER_SO_read()==1)
    c = c |0x10000000;

c = c >> (n*4);
masc = ~(masc >> (n*4));

CR.write((CR.read() & masc) |c);

};

//!Instruction cntlzw behavior method.
// Count Leading Zeros Word
void ac_behavior( cntlzw )
{
    dbg_printf(" cntlzw%s r%d, r%d\n\n", (rc==1?" ":""),ra,rs);

    unsigned int urs=GPR[rs];
    unsigned int masc=0x80000000;
    unsigned int n=0;

    while(n < 32) {
        if(urs & masc)
            break;

        n++;
        masc=masc>>1;
    }

    GPR[ra] = n;

```

```
    if (rc==1)
        CR0_update(n);
};

//!Instruction crand behavior method.
// Condition Register AND
void ac_behavior( crand )
{
    dbg_printf(" crand %d, %d, %d\n\n",bt,ba,bb);

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba=CR.read();
    CRbb=CR.read();

    /* Shift source bit to first position and zero another */
    CRba=(CRba<<ba)&0x80000000;
    CRbb=(CRbb<<bb)&0x80000000;

    CRbt=(CRba & CRbb) & 0x80000000;
    CRbt=CRbt>>bt;

    if(CRbt)
        CR.write(CR.read() |CRbt);
    else
        CR.write(CR.read() & ~(0x80000000 >> bt));
};

//!Instruction crandc behavior method.
// Condition Register AND with Complement
void ac_behavior( crandc )
{
    dbg_printf(" crandc %d, %d, %d\n\n",bt,ba,bb);
```

```

unsigned int CRbt;
unsigned int CRba;
unsigned int CRbb;

CRba=CR.read();
CRbb=CR.read();

/* Shift source bit to first position and zero another */
CRba=(CRba<<ba)&0x80000000;
CRbb=(CRbb<<bb)&0x80000000;

CRbt=(CRba & ~CRbb) & 0x80000000;
CRbt=CRbt>>bt;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> bt));
};

//!Instruction creqv behavior method.
// Condition Register Equivalent
void ac_behavior( creqv )
{
    dbg_printf(" creqv %d, %d, %d\n\n",bt,ba,bb);

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba=CR.read();
    CRbb=CR.read();

    /* Shift source bit to first position and zero another */

```



```

CRba=(CRba<<ba)&0x80000000;
CRbb=(CRbb<<bb)&0x80000000;

CRbt=~(CRba ^CRbb) & 0x80000000;
CRbt=CRbt>>bt;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> bt));
};

//!Instruction crnand behavior method.
// Condition Register NAND
void ac_behavior( crnand )
{
    dbg_printf(" crnand %d, %d, %d\n\n",bt,ba,bb);

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba=CR.read();
    CRbb=CR.read();

    /* Shift source bit to first position and zero another */
    CRba=(CRba<<ba)&0x80000000;
    CRbb=(CRbb<<bb)&0x80000000;

    CRbt=~(CRba & CRbb) & 0x80000000;
    CRbt=CRbt>>bt;

    if(CRbt)
        CR.write(CR.read() |CRbt);
    else

```

```

        CR.write(CR.read() & ~(0x80000000 >> bt));
};

//!Instruction crnor behavior method.
// Condition Register NOR
void ac_behavior( crnor )
{
    dbg_printf(" crnor %d, %d, %d\n\n",bt,ba,bb);

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba=CR.read();
    CRbb=CR.read();

    /* Shift source bit to first position and zero another */
    CRba=(CRba<<ba)&0x80000000;
    CRbb=(CRbb<<bb)&0x80000000;

    CRbt=~(CRba |CRbb) & 0x80000000;
    CRbt=CRbt>>bt;

    if(CRbt)
        CR.write(CR.read() |CRbt);
    else
        CR.write(CR.read() & ~(0x80000000 >> bt));
};

//!Instruction cror behavior method.
// Condition Register OR
void ac_behavior( cror )
{
    dbg_printf(" cror %d, %d, %d\n\n",bt,ba,bb);

```

```

unsigned int CRbt;
unsigned int CRba;
unsigned int CRbb;

CRba=CR.read();
CRbb=CR.read();

/* Shift source bit to first position and zero another */
CRba=(CRba<<ba)&0x80000000;
CRbb=(CRbb<<bb)&0x80000000;

CRbt=(CRba | CRbb) & 0x80000000;
CRbt=CRbt>>bt;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> bt));
};

//!Instruction crorc behavior method.
// Condition Register OR with Complement
void ac_behavior( crorc )
{
    dbg_printf(" crorc %d, %d, %d\n\n",bt,ba,bb);

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba=CR.read();
    CRbb=CR.read();

    /* Shift source bit to first position and zero another */
    CRba=(CRba<<ba)&0x80000000;

```

```

CRbb=(CRbb<<bb)&0x80000000;

CRbt=(CRba |~CRbb) & 0x80000000;
CRbt=CRbt>>bt;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> bt));

};

//!Instruction crxor behavior method.
// Condition Register XOR
void ac_behavior( crxor )
{
    dbg_printf(" crxor %d, %d, %d\n\n",bt,ba,bb);

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba=CR.read();
    CRbb=CR.read();

    /* Shift source bit to first position and zero another */
    CRba=(CRba<<ba)&0x80000000;
    CRbb=(CRbb<<bb)&0x80000000;

    CRbt=(CRba ^CRbb) & 0x80000000;
    CRbt=CRbt>>bt;

    if(CRbt)
        CR.write(CR.read() |CRbt);
    else

```

```

        CR.write(CR.read() & ~(0x80000000 >> bt));
};

//!Instruction divw behavior method.
// Divide Word
void ac_behavior( divw )
{
    dbg_printf(" divw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r
    int result=(int)GPR[ra]/(int)GPR[rb];

    if (oe==1)
        divws_XER_OV_SO_update(result,GPR[ra],GPR[rb]);

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction divwu behavior method.
// Divide Word Unsigned
void ac_behavior( divwu )
{
    dbg_printf(" divwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r
    unsigned int result=(unsigned int)GPR[ra]/(unsigned int)GPR[rb];

    if (oe==1)
        divwu_XER_OV_SO_update(result,GPR[ra],GPR[rb]);

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

```

```
};

//!Instruction eqv behavior method.
// Equivalent
void ac_behavior( eqv )
{
    dbg_printf(" eqv%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);

    GPR[ra] = ~(GPR[rs] ^GPR[rb]);

    if (rc==1)
        CR0_update(GPR[ra]);
};

//!Instruction extsb behavior method.
// Extend Sign Byte
void ac_behavior( extsb )
{
    dbg_printf(" extsb%s r%d, r%d\n\n", (rc==1?" ":""), ra, rs);

    GPR[ra] = (char)(GPR[rs]);

    if (rc==1)
        CR0_update(GPR[ra]);
};

//!Instruction extsh behavior method.
// Extend Sign Halfword
void ac_behavior( extsh )
{
    dbg_printf(" extsh%s r%d, r%d\n\n", (rc==1?" ":""), ra, rs);

    GPR[ra] = (short int)(GPR[rs]);

    if (rc==1)
```

```
        CR0_update(GPR[ra]);
};

//!Instruction lbz behavior method.
// Load Byte and Zero
void ac_behavior( lbz )
{
    dbg_printf(" lbz r%d, %d(r%d)\n\n",rt,d,ra);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    GPR[rt] = (unsigned int)MEM.read_byte(ea);
};

//!Instruction lbzu behavior method.
// Load Byte and Zero with Update
void ac_behavior( lbzu )
{
    dbg_printf(" lbzu r%d, %d(r%d)\n\n",rt,d,ra);

    int ea;

    ea=GPR[ra]+(short int)d;

    GPR[ra] = ea;
    GPR[rt] = (unsigned int)MEM.read_byte(ea);
};

//!Instruction lbzux behavior method.
// Load Byte and Zero with Update Indexed
```

```
void ac_behavior( lbzux )
{
    dbg_printf(" lbzux r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea;

    ea=GPR[ra]+GPR[rb];

    GPR[ra] = ea;
    GPR[rt] = (unsigned int)MEM.read_byte(ea);
};

//!Instruction lbzx behavior method.
// Load Byte and Zero Indexed
void ac_behavior( lbzx )
{
    dbg_printf(" lbzx r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];

    GPR[rt] = (unsigned int)MEM.read_byte(ea);
};

//!Instruction lha behavior method.
// Load Halfword Algebraic
void ac_behavior( lha )
{
    dbg_printf(" lha r%d, %d(r%d)\n\n",rt,d,ra);

    int ea;
```



```
    if(ra!=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    GPR[rt] = (short int)MEM.read_half(ea);
};

//!Instruction lhau behavior method.
// Load Halfword Algebraic with Update
void ac_behavior( lhau )
{
    dbg_printf(" lhau r%d, %d(r%d)\n\n",rt,d,ra);

    int ea=GPR[ra]+(short int)d;

    GPR[ra] = ea;
    GPR[rt] = (short int)MEM.read_half(ea);

};

//!Instruction lhaux behavior method.
// Load Halfword Algebraic with Update Indexed
void ac_behavior( lhaux )
{
    dbg_printf(" lhaux r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea=GPR[ra]+GPR[rb];

    GPR[ra] = ea;
    GPR[rt] = (short int)MEM.read_half(ea);

};
```

```
//!Instruction lhax behavior method.
// Load Halfword Algebraic Indexed
void ac_behavior( lhax )
{
    dbg_printf(" lhax r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea;

    if(ra !=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];

    GPR[rt] = (short int)MEM.read_half(ea);

};

//!Instruction lhbrx behavior method.
// Load Halfword Byte-Reverse Indexed
void ac_behavior( lhbrx )
{
    dbg_printf(" lhbrx r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea;

    if(ra !=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];

    GPR[rt] = (((int)(MEM.read_byte(ea+1)) & 0x000000FF)<<8) |((int)(MEM.read_byte(
& 0x000000FF));

};
```

```
//!Instruction lhz behavior method.
// Load Halfword and Zero
void ac_behavior( lhz )
{
    dbg_printf(" lhz r%d, %d(r%d)\n\n",rt,d,ra);

    int ea;

    if(ra !=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    GPR[rt] = (unsigned short int)MEM.read_half(ea);

};

//!Instruction lhzu behavior method.
// Load Halfword and Zero with Update
void ac_behavior( lhzu )
{
    dbg_printf(" lhzu r%d, %d(%d)\n\n",rt,d,ra);

    int ea=GPR[ra]+(short int)d;

    GPR[ra] = ea;
    GPR[rt] = (unsigned short int)MEM.read_half(ea);
};

//!Instruction lhzux behavior method.
// Load Halfword and Zero with Update Indexed
void ac_behavior( lhzux )
{
    dbg_printf(" lhzux r%d, r%d, r%d\n\n",rt,ra,rb);
```

```
int ea=GPR[ra]+GPR[rb];

GPR[ra] = ea;
GPR[rt] = (unsigned short int)MEM.read_half(ea);
};

//!Instruction lhzx behavior method.
// Load Halfword and Zero Indexed
void ac_behavior( lhzx )
{
    dbg_printf(" lhzx r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea;

    if(ra !=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];

    GPR[rt] = (unsigned short int)MEM.read_half(ea);
};

//!Instruction lmw behavior method.
// Load Multiple Word
void ac_behavior( lmw )
{
    dbg_printf(" lmw r%d, %d(r%d)\n\n",rt,d,ra);

    int ea;
    unsigned int r;

    if(ra !=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;
```

```
r=rt;

while(r<=31) {
    if((r!=ra)||(r==31))
        GPR[r] = MEM.read(ea);
    r=r+1;
    ea=ea+4;
}
};

//!Instruction lswi behavior method.
// Load String Word Immediate
void ac_behavior( lswi )
{
    dbg_printf(" lswi r%d, r%d, %d\n\n",rt,ra,nb);

    int ea;
    unsigned int cnt,n;
    unsigned int rfinal,r;
    unsigned int i,masc;

    if(ra!=0)
        ea=GPR[ra];
    else
        ea=0;

    if(nb==0)
        cnt=32;
    else
        cnt=nb;

    n=cnt;
    rfinal=((rt + ceil(cnt,4) - 1) % 32);
    r=rt-1;
```

```

i=0;

while(n>0) {
    if(i==0) {
        r=r+1;
        if(r==32)
            r=0;
        if((r!=ra)||(r==rfinal))
            GPR[r] = 0;
    }
    if((r!=ra)||(r==rfinal)) {
        masc=0xFF000000>>i;
        masc=~masc;
        GPR[r] = (GPR[r] & masc);
        GPR[r] = (((unsigned int)MEM.read_byte(ea)) << (24-i)) |GPR[r];
    }
    i=i+8;
    if(i==32)
        i=0;
    ea=ea+1;
    n=n-1;
}
};

//!Instruction lswx behavior method.
// Load String Word Indexed
void ac_behavior( lswx )
{
    dbg_printf(" lswx r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea;
    unsigned int cnt,n;
    unsigned int rfinal,r;
    unsigned int i,masc;

```

```

if(ra!=0)
    ea=GPR[ra]+GPR[rb];
else
    ea=GPR[rb];

cnt=XER_TBC_read();
n=cnt;
rfinal=((rt + ceil(cnt,4) - 1) % 32);
r=rt-1;
i=0;

while(n>0) {
    if(i==0) {
        r=r+1;
        if(r==32)
            r=0;
        if(((r!=ra) && (r!=rb)) ||(r==rfinal))
            GPR[r] = 0;
    }
    if(((r!=ra) && (r!=rb)) ||(r==rfinal)) {
        masc=0xFF000000>>i;
        masc=~masc;
        GPR[r] = (GPR[r] & masc);
        GPR[r] = (((unsigned int)MEM.read_byte(ea)) << (24-i)) |GPR[r];
    }
    i=i+8;
    if(i==32)
        i=0;
    ea=ea+1;
    n=n-1;
}
};

//!Instruction lwbrx behavior method.

```

```
// Load Word Byte-Reverse Indexed
void ac_behavior( lwbrx )
{
    dbg_printf(" lwbrx r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];

    GPR[rt] = (((unsigned int)MEM.read_byte(ea+3) & 0x000000FF) << 24) |
              (((unsigned int)MEM.read_byte(ea+2) & 0x000000FF) << 16) |
              (((unsigned int)MEM.read_byte(ea+1) & 0x000000FF) << 8) |
              ((unsigned int)MEM.read_byte(ea) & 0x000000FF);

};

//!Instruction lwz behavior method.
// Load Word and Zero
void ac_behavior( lwz )
{
    dbg_printf(" lwz r%d, %d(r%d)\n\n",rt,d,ra);

    int ea;

    if(ra !=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    GPR[rt] = MEM.read(ea);
};
```



```
//!Instruction lwzu behavior method.
// Load Word and Zero with Update
void ac_behavior( lwzu )
{
    dbg_printf(" lwzu r%d, %d(r%d)\n\n",rt,d,ra);

    int ea=GPR[ra]+(short int)d;

    GPR[ra] = ea;
    GPR[rt] = MEM.read(ea);
};

//!Instruction lwzux behavior method.
// Load Word and Zero with Update Indexed
void ac_behavior( lwzux )
{
    dbg_printf(" lwzux r%d, r%d, r%d\n\n",rt,ra,rb);

    int ea=GPR[ra]+GPR[rb];

    GPR[ra] = ea;
    GPR[rt] = MEM.read(ea);
};

//!Instruction lwzx behavior method.
// Load Word and Zero Indexed
void ac_behavior( lwzx )
{
    dbg_printf(" lwzx %d, %d, %d\n\n",rt,ra,rb);

    int ea;

    if(ra !=0)
        ea=GPR[ra]+GPR[rb];
    else
```

```

    ea=GPR[rb];

    GPR[rt] = MEM.read(ea);
};

/*****/

//!Instruction macchw behavior method.
// Multiply Accumulate Cross Halfword to Word Modulo Signed
void ac_behavior( macchw )
{
    dbg_printf(" macchw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")))

    genericMac ( NORMAL , CROSS , MODULE , SIGNED , oe , rc , rt , ra , rb);

};

//!Instruction macchws behavior method.
// Multiply Accumulate Cross Halfword to Word Saturate Signed
void ac_behavior( macchws )
{
    dbg_printf(" macchws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")))

    genericMac ( NORMAL , CROSS , SATURATE , SIGNED , oe , rc , rt , ra , rb
);

};

//!Instruction macchwsu behavior method.
// Multiply Accumulate Cross Halfword to Word Saturate Unsigned
void ac_behavior( macchwsu )
{
    dbg_printf(" macchwsu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")))

    genericMac ( NORMAL , CROSS , SATURATE , UNSIGNED , oe , rc , rt , ra ,

```

```

rb );

};

//!Instruction macchwu behavior method.
// Multiply Accumulate Cross Halfword to Word Modulo Unsigned
void ac_behavior( macchwu )
{
    dbg_printf(" macchwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
    genericMac ( NORMAL , CROSS , MODULE , UNSIGNED , oe , rc , rt , ra , rb
);
};

//!Instruction machhw behavior method.
// Multiply Accumulate High Halfword to Word Modulo Signed
void ac_behavior( machhw )
{
    dbg_printf(" machhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
    genericMac ( NORMAL , HIGH , MODULE , SIGNED , oe , rc , rt , ra , rb );
};

//!Instruction machhws behavior method.
// Multiply Accumulate High Halfword to Word Saturate Signed
void ac_behavior( machhws )
{
    dbg_printf(" machhws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
    genericMac ( NORMAL , HIGH , SATURATE , SIGNED , oe , rc , rt , ra , rb
);
};

```

```

//!Instruction machhwsu behavior method.
// Multiply Accumulate High Halfword to Word Saturate Unsigned
void ac_behavior( machhwsu )
{
    dbg_printf(" machhwsu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

    genericMac ( NORMAL , HIGH , SATURATE , UNSIGNED , oe , rc , rt , ra ,
rb );

};

//!Instruction machhwu behavior method.
// Multiply Accumulate High Halfword to Word Modulo Unsigned
void ac_behavior( machhwu )
{
    dbg_printf(" machhwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

    genericMac ( NORMAL , HIGH , MODULE , UNSIGNED , oe , rc , rt , ra , rb
);

};

//!Instruction maclhw behavior method.
// Multiply Accumulate Low Halfword to Word Modulo Signed
void ac_behavior( maclhw )
{
    dbg_printf(" maclhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

    genericMac ( NORMAL , LOW , MODULE , SIGNED , oe , rc , rt , ra , rb );

};

//!Instruction maclhws behavior method.
// Multiply Accumulate Low Halfword to Word Saturate Signed

```

```

void ac_behavior( maclhws )
{
    dbg_printf(" maclhws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

    genericMac ( NORMAL , LOW , SATURATE , SIGNED , oe , rc , rt , ra , rb
);

};

//!Instruction maclhwsu behavior method.
// Multiply Accumulate Low Halfword to Word Saturate Unsigned
void ac_behavior( maclhwsu )
{
    dbg_printf(" maclhwsu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

    genericMac ( NORMAL , LOW , SATURATE , UNSIGNED , oe , rc , rt , ra , rb
);

};

//!Instruction maclhwu behavior method.
// Multiply Accumulate Low Halfword to Word Modulo Unsigned
void ac_behavior( maclhwu )
{
    dbg_printf(" maclhwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

    genericMac ( NORMAL , LOW , MODULE , UNSIGNED , oe , rc , rt , ra , rb
);

};

/*****/

//!Instruction mcrf behavior method.

```

```
// Move Condition Register Field
void ac_behavior( mcrf )
{
    dbg_printf(" mcrf %d, %d\n\n",bf,bfa);

    unsigned int m=bfa;
    unsigned int n=bf;
    unsigned int tmp,i;
    unsigned int masc;
    /* n <- m */

    /* Clean all bits except m-bits in tmp */
    masc=0xF0000000;
    for(i=0 ; i<m ; i++)
        masc=masc>>4;
    tmp=CR.read();
    tmp=tmp & masc;

    /* Put m-bits in n position */
    if(n<m)
        for(i=0 ; i < m-n ; i++)
            tmp=tmp<<4;
    else
        if(n>m) /* Else nothing */
            for(i=0 ; i < n-m ; i++)
                tmp=tmp>>4;

    /* Clean all bits of n-bits and make an or with tmp */
    masc=0xF0000000;
    for(i=0 ; i<n ; i++)
        masc=masc>>4;
    masc=~masc;
    CR.write((CR.read() & masc)|tmp);
};
```

```
//!Instruction mcrxr behavior method.
// Move to Condition Register from XER
void ac_behavior( mcrxr )
{
    dbg_printf(" mcrxr %d\n\n",bf);

    unsigned int n=bf;
    unsigned int i;
    unsigned int tmp=0x00;

    /* Calculate tmp bits by XER */
    if(XER_SO_read())
        tmp=tmp |0x80000000;
    if(XER_OV_read())
        tmp=tmp |0x40000000;
    if(XER_CA_read())
        tmp=tmp |0x20000000;

    /* Move altered bits to correct CR field */
    for(i=0 ; i<n ; i++)
        tmp=tmp>>4;
    CR.write(tmp);

    /* Clean XER bits */
    XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    XER.write(XER.read() & 0x7FFFFFFF); /* Write 0 to bit 0 SO */
    XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */
};

//!Instruction mfcr behavior method.
// Move From Condition Register
void ac_behavior( mfcr )
{
    dbg_printf(" mfcr r%d\n\n",rt);
```

```
GPR[rt] = CR.read();
};

//!Instruction mfspr behavior method.
// Move From Special Purpose Register
void ac_behavior( mfspr )
{
    /* This instruction is a fix, other implementations can be better */
    dbg_printf(" mfspr r%d,%d\n\n",rt,sprf);
    unsigned int spvalue=sprf;
    spvalue=((spvalue>>5) & 0x0000001f ) |
        ((spvalue<<5) & 0x000003e0 );

    switch(spvalue) {

        /* LR */
        case 0x008:
            GPR[rt] = LR.read();
            break;

        /* CTR */
        case 0x009:
            GPR[rt] = CTR.read();
            break;

        /* USPRG0 */
        case 0x100:
            GPR[rt] = USPRG0.read();
            break;

        /* SPRG4 */
        case 0x104:
            GPR[rt] = SPRG4.read();
```



```
        break;

/* SPRG5 */
case 0x105:
    GPR[rt] = SPRG5.read();
    break;

/* SPRG6 */
case 0x106:
    GPR[rt] = SPRG6.read();
    break;

/* SPRG7 */
case 0x107:
    GPR[rt] = SPRG7.read();
    break;

/* Not implemented yet! */
default:
    dbg_printf("\nERROR!\n");
    exit(-1);
    break;
}
};

//!Instruction mtcrrf behavior method.
// Move to Condition Register Fields
void ac_behavior( mtcrrf ) {
    dbg_printf(" mtcrrf %d, r%d\n\n",xfm,rs);

    unsigned int tmpop,tmpmask;
```

```

unsigned int mask;
unsigned int i;
unsigned int crm=xfm;

tmpmask=0xF0000000;
tmpop=0x80;
mask=0;
for(i=0;i<8;i++) {
    if(crm & tmpop)
        mask=mask |tmpmask;
    tmpmask=tmpmask>>4;
    tmpop=tmpop>>1;
}

CR.write((GPR[rs] & mask) |(CR.read() & ~mask));
};

/*!Instruction mtspr behavior method.
// Move To Special Purpose Register
void ac_behavior( mtspr )
{
    /* This instruction is a fix, other implementations can be better */
    dbg_printf(" mtspr %d,r%d\n\n",sprf,rs);
    unsigned int spvalue=sprf;
    spvalue=((spvalue>>5) & 0x0000001f ) |
        ((spvalue<<5) & 0x000003e0 );

    switch(spvalue) {

        /* LR */
        case 0x008:
            LR.write(GPR[rs]);
            break;

```

```
/* CTR */
case 0x009:
    CTR.write(GPR[rs]);
    break;

/* USPRG0 */
case 0x100:
    USPRG0.write(GPR[rs]);
    break;

/* SPRG4 */
case 0x104:
    SPRG4.write(GPR[rs]);
    break;

/* SPRG5 */
case 0x105:
    SPRG5.write(GPR[rs]);
    break;

/* SPRG6 */
case 0x106:
    SPRG6.write(GPR[rs]);
    break;

/* SPRG7 */
case 0x107:
    SPRG7.write(GPR[rs]);
    break;

/* Not implemented yet! */
default:
    dbg_printf("\nERROR!\n");
```

```

        exit(-1);
        break;

    }

};

/*****

//!Instruction mulchw behavior method.
// Multiply Cross Halfword to Word Signed
void ac_behavior( mulchw )
{
    dbg_printf(" mulchw%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    genericMac ( MULTIPLY , CROSS , MODULE , SIGNED , 0 , rc , rt , ra , rb
);

};

//!Instruction mulchwu behavior method.
// Multiply Cross Halfword to Word Unsigned
void ac_behavior( mulchwu )
{
    dbg_printf(" mulchwu%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    genericMac ( MULTIPLY , CROSS , MODULE , UNSIGNED , 0 , rc , rt , ra ,
rb );

};

//!Instruction mulhhw behavior method.
// Multiply High Halfword to Word Signed
void ac_behavior( mulhhw )

```

```

{
    dbg_printf(" mulhhw%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    genericMac ( MULTIPLY , HIGH , MODULE , SIGNED , 0 , rc , rt , ra , rb
);
};

//!Instruction mulhhwu behavior method.
// Multiply High Halfword to Word Unsigned
void ac_behavior( mulhhwu )
{
    dbg_printf(" mulhhwu%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    genericMac ( MULTIPLY , HIGH , MODULE , UNSIGNED , 0 , rc , rt , ra , rb
);
};

/*****/

//!Instruction mulhw behavior method.
// Multiply High Word
void ac_behavior( mulhw )
{
    dbg_printf(" mulhw%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    int high;
    long long int prod =
        (long long int)(int)GPR[ra]*(long long int)(int)GPR[rb];

    unsigned long long int shprod=prod;
    shprod=shprod>>32;
    high=shprod;
}

```

```

    GPR[rt] = high;

    if (rc==1)
        CR0_update(high);
};

//!Instruction mulhwu behavior method.
// Multiply High Word Unsigned
void ac_behavior( mulhwu )
{
    dbg_printf(" mulhwu%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    unsigned int high;
    unsigned long long int prod =
        (unsigned long long int)(unsigned int)GPR[ra]*
        (unsigned long long int)(unsigned int)GPR[rb];

    prod=prod>>32;
    high=prod;

    GPR[rt] = high;

    if (rc==1)
        CR0_update(high);
};

/*****/
//!Instruction mullhw behavior method.
// Multiply Low Halfword to Word Signed
void ac_behavior( mullhw )
{
    dbg_printf(" mullhw%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

```

```

    genericMac ( MULTIPLY , LOW , MODULE , SIGNED , 0 , rc , rt , ra , rb );

};

//!Instruction mullhwu behavior method.
// Multiply Low Halfword to Word Unsigned
void ac_behavior( mullhwu )
{
    dbg_printf(" mullhwu%s r%d, r%d, r%d\n\n", (rc==1?" ":""),rt,ra,rb);

    genericMac ( MULTIPLY , LOW , MODULE , UNSIGNED , 0 , rc , rt , ra , rb
);

};

/*****/

//!Instruction mulli behavior method.
// Multiply Low Immediate
void ac_behavior( mulli )
{
    dbg_printf(" mulli r%d, r%d, %d\n\n",rt,ra,d);

    long long int prod =
        (long long int)(int)GPR[ra] * (long long int)(short int)(d);

    int low;
    low=prod;

    GPR[rt] = low;

};

//!Instruction mullw behavior method.
// Multiply Low Word

```

```

void ac_behavior( mullw )
{
    dbg_printf(" mullw%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    int low;
    long long int prod =
        (long long int)(int)GPR[ra]*(long long int)(int)GPR[rb];

    low=prod;

    GPR[rt] = low;

    if (oe==1) {
        if(prod != (long long int)(int)low) {
            XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
            XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
        }
        else
            XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    }

    if (rc==1)
        CR0_update(low);
};

//!Instruction nand behavior method.
// NAND
void ac_behavior( nand )
{
    dbg_printf(" nand%s r%d, r%d, r%d\n\n", (rc==1?".":""),ra,rs,rb);

    int result=~(GPR[rs] & GPR[rb]);

    GPR[ra] = result;

```



```

    if (rc==1)
        CR0_update(result);

};

//!Instruction neg behavior method.
// Negate
void ac_behavior( neg )
{
    dbg_printf(" neg%s r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), rt, ra);

    long long int longresult = ~((unsigned int)GPR[ra])+1;
    int result = ~(GPR[ra]) + 1;

    if (oe==1) {
        if(longresult != (long long int)result) {
            XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
            XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
        }
        else
            XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    }

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

/***** 405 instructions *****/

//!Instruction nmacchw behavior method.
// Negative Multiply Accumulate Cross Halfword to Word Modulo Signed
void ac_behavior( nmacchw )
{

```

```

    dbg_printf(" nmacchw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
);

genericMac ( NEGATIVE , CROSS , MODULE , SIGNED , oe , rc , rt , ra , rb
);

};

//!Instruction nmacchws behavior method.
// Negative Multiply Accumulate Cross Halfword to Word Saturate Signed
void ac_behavior( nmacchws )
{
    dbg_printf(" nmacchws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
);

    genericMac ( NEGATIVE , CROSS , SATURATE , SIGNED , oe , rc , rt , ra ,
rb );
};

//!Instruction nmachhw behavior method.
// Negative Multiply Accumulate High Halfword to Word Modulo Signed
void ac_behavior( nmachhw )
{
    dbg_printf(" nmachhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
);

    genericMac ( NEGATIVE , HIGH , MODULE , SIGNED , oe , rc , rt , ra , rb
);
};

//!Instruction nmachhws behavior method.
// Negative Multiply Accumulate High Halfword to Word Saturate Signed
void ac_behavior( nmachhws )
{
    dbg_printf(" nmachhws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
);
};

```

```

    genericMac ( NEGATIVE , HIGH , SATURATE , SIGNED , oe , rc , rt , ra ,
rb );

};

//!Instruction nmaclhw behavior method.
// Negative Multiply Accumulate Low Halfword to Word Modulo Signed
void ac_behavior( nmaclhw )
{
    dbg_printf(" nmaclhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
    genericMac ( NEGATIVE , LOW , MODULE , SIGNED , oe , rc , rt , ra , rb
);

};

//!Instruction nmaclhws behavior method.
// Negative Multiply Accumulate Low Halfword to Word Saturate Signed
void ac_behavior( nmaclhws )
{
    dbg_printf(" nmaclhws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))
    genericMac ( NEGATIVE , LOW , SATURATE , SIGNED , oe , rc , rt , ra , rb
);

};

/*****/

//!Instruction nor behavior method.
// NOR
void ac_behavior( nor )
{
    dbg_printf(" nor%s r%d, r%d, r%d\n\n", (rc==1?".":""),ra,rs,rb);

```

```

    int result = ~(GPR[rs] | GPR[rb]);

    GPR[ra] = result;

    if (rc==1)
        CR0_update(result);
};

//!Instruction ore behavior method.
// OR
void ac_behavior( ore )
{
    dbg_printf(" ore%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);

    int result = GPR[rs] | GPR[rb];

    GPR[ra] = result;

    if (rc==1)
        CR0_update(result);
};

//!Instruction orc behavior method.
// OR with Complement
void ac_behavior( orc )
{
    dbg_printf(" orc%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);

    int result = GPR[rs] | ~GPR[rb];

    GPR[ra] = result;

    if (rc==1)
        CR0_update(result);
};

```

```
//!Instruction ori behavior method.
// OR Immediate
void ac_behavior( ori )
{
    dbg_printf(" ori r%d, r%d, %d\n\n",ra,rs,ui);

    GPR[ra] = GPR[rs] |(int)((unsigned short int)ui);
};

//!Instruction oris behavior method.
// OR Immediate Shifted
void ac_behavior( oris )
{
    dbg_printf(" oris r%d, r%d, r%d\n\n",ra,rs,ui);

    GPR[ra] = GPR[rs] |(((int)((unsigned short int)ui)) << 16);
};

//!Instruction rlwimi behavior method.
// Rotate Left Word Immediate then Mask Insert
void ac_behavior( rlwimi )
{
    dbg_printf(" rlwimi%s r%d, r%d, %d, %d, %d\n\n", (rc==1?" ":""),ra,rs,sh,mb,me);

    unsigned int r=rotr(GPR[rs],sh);
    unsigned int m=mask32rlw(mb,me);

    GPR[ra] = (r & m) |(GPR[ra] & ~m);

    if (rc==1)
        CR0_update(GPR[ra]);
};
```

```

//!Instruction rlwinm behavior method.
// Rotate Left Word Immediate then AND with Mask
void ac_behavior( rlwinm )
{
    dbg_printf(" rlwinm%s r%d, r%d, %d, %d, %d\n\n", (rc==1?".":""), ra, rs, sh, mb, me);

    unsigned int r=rotl(GPR[rs], sh);
    unsigned int m=mask32rlw(mb, me);

    GPR[ra] = (r & m);

    if (rc==1)
        CR0_update(GPR[ra]);
};

//!Instruction rlwnm behavior method.
// Rotate Left Word then AND with Mask
void ac_behavior( rlwnm )
{
    dbg_printf(" rlwnm%s r%d, r%d, %d, %d, %d\n\n", (rc==1?".":""), ra, rs, rb, mb, me);

    unsigned int r=rotl(GPR[rs], (GPR[rb] | 0x0000001F));
    unsigned int m=mask32rlw(mb, me);

    GPR[ra] = (r & m);

    if (rc==1)
        CR0_update(GPR[ra]);
};

//!Instruction sc behavior method.
// System Call
/* The registers used in this intruction may be defined better */
void ac_behavior( sc )

```

```

{
    dbg_printf(" sc\n\n");

    SRR1.write(MSR.read()); /* It uses a function to join MSR fields */

    /* Write WE, EE, PR, DR and IR as 0 in MSR */
    MSR.write(MSR.read() |0xFFFB3FCF);

    SRR0.write((ac_pc-4)+4); /* Only to understand pre-increment */

    ac_pc=((EVPR.read() & 0xFFFF0000) |0x00000C00);

};

//!Instruction slw behavior method.
// Shift Left Word
void ac_behavior( slw )
{
    dbg_printf(" slw%s r%d, r%d, r%d\n\n", (rc==1?" ":""),ra,rs,rb);

    unsigned int n=(GPR[rb] & 0x0000001F);
    unsigned int r=rotr(GPR[rs],n);
    unsigned int m;

    if((GPR[rb] & 0x00000020)==0x00) /* Check bit 26 */
        m=mask32rlw(0,31-n);
    else
        m=0;

    int result=r & m;
    GPR[ra] = result;

    if (rc==1)
        CR0_update(result);
}

```

```

};

//!Instruction sraw behavior method.
// Shift Right Algebraic Word
void ac_behavior( sraw )
{
    dbg_printf(" sraw%s r%d, r%d, r%d\n\n", (rc==1?".":""), ra, rs, rb);

    unsigned int n=(GPR[rb] & 0x0000001F);
    unsigned int r=rotr(GPR[rs],32-n);
    unsigned int m;
    unsigned int s;

    if((GPR[rb] & 0x00000020)==0x00) /* Check bit 26 */
        m=mask32rlw(n,31);
    else
        m=0;

    s=(GPR[rs] & 0x80000000);
    if(s!=0)
        s=0xFFFFFFFF;

    /* Write ra */
    int result=(r & m) |(s & ~m);

    GPR[ra] = result;

    /* Update XER CA */
    if(s && ((r & ~m)!=0))
        XER.write(XER.read() |0x20000000); /* Write 1 to bit 2 CA */
    else
        XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */

    /* Update CR register */
    if (rs==1)

```



```

        CR0_update(result);

};

//!Instruction srawi behavior method.
// Shift Right Algebraic Word Immediate
void ac_behavior( srawi )
{
    dbg_printf(" srawi%s r%d, r%d, %d\n\n", (rc==1?".":""), ra, rs, sh);

    unsigned int n=sh;
    unsigned int r=rotr(GPR[rs], 32-n);
    unsigned int m=mask32rlw(n, 31);
    unsigned int s=(GPR[rs] & 0x80000000);

    if(s!=0)
        s=0xFFFFFFFF;

    int result=(r & m) |(s & ~m);

    GPR[ra] = result;

    /* Update XER CA */
    if(s && ((r & ~m)!=0))
        XER.write(XER.read() |0x20000000); /* Write 1 to bit 2 CA */
    else
        XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */

    if (rc==1)
        CR0_update(result);
};

//!Instruction srw behavior method.
// Shift Right Word
void ac_behavior( srw )

```

```
{
    dbg_printf(" srw%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);

    unsigned int n=(GPR[rb] & 0x0000001F);
    unsigned int r=rotr(GPR[rs],32-n);
    unsigned int m;

    if((GPR[rb] & 0x00000020)==0x00) /* Check bit 26 */
        m=mask32rlw(n,31);
    else
        m=0;

    int result=r & m;

    GPR[ra] = result;

    if (rc==1)
        CR0_update(result);
};

//!Instruction stb behavior method.
// Store Byte
void ac_behavior( stb )
{
    dbg_printf(" stb r%d, %d(r%d)\n\n",rs,d,ra);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    MEM.write_byte(ea,(unsigned char)GPR[rs]);
};
```

```
//!Instruction stbu behavior method.
// Store Byte with Update
void ac_behavior( stbu )
{
    dbg_printf(" stbu r%d, %d(r%d)\n\n",rs,d,ra);

    int ea=GPR[ra]+(short int)d;

    MEM.write_byte(ea,(unsigned char)GPR[rs]);
    GPR[ra] = ea;
};

//!Instruction stbux behavior method.
// Store Byte with Update Indexed
void ac_behavior( stbux )
{
    dbg_printf(" stbux r%d, r%d, r%d\n\n",rs,ra,rb);

    int ea=GPR[ra]+GPR[rb];

    MEM.write_byte(ea,(unsigned char)GPR[rs]);
    GPR[ra] = ea;
};

//!Instruction stbx behavior method.
// Store Byte Indexed
void ac_behavior( stbx )
{
    dbg_printf(" stbx r%d, r%d, r%d\n\n",rs,ra,rb);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+GPR[rb];
```

```
else
    ea=GPR[rb];

MEM.write_byte(ea,(unsigned char)GPR[rs]);
};

//!Instruction sth behavior method.
// Store Halfword
void ac_behavior( sth )
{
    dbg_printf(" sth r%d, %d(r%d)\n\n",rs,d,ra);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    MEM.write_half(ea,(unsigned short int)GPR[rs]);
};

//!Instruction sthbrx behavior method.
// Store Halfword Byte-Reverse Indexed
void ac_behavior( sthbrx )
{
    dbg_printf(" sthbrx r%d, r%d, r%d\n\n",rs,ra,rb);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];
```

```

MEM.write_half(ea,(unsigned short int)
    ( ((GPR[rs] & 0x000000FF) << 8) |
      ((GPR[rs] & 0x0000FF00) >> 8) ));
}

//!Instruction sthu behavior method.
// Store Halfword with Update
void ac_behavior( sthu )
{
    dbg_printf(" sthu r%d, %d(r%d)\n\n",rs,d,ra);

    int ea=GPR[ra]+(short int)d;

    MEM.write_half(ea,(unsigned short int)GPR[rs]);
    GPR[ra] = ea;
};

//!Instruction sthux behavior method.
// Store Halfword with Update Indexed
void ac_behavior( sthux )
{
    dbg_printf("sthux r%d, r%d, r%d\n\n",rs,ra,rb);

    int ea=GPR[ra]+GPR[rb];

    MEM.write_half(ea,(unsigned short int)GPR[rs]);
    GPR[ra] = ea;
};

//!Instruction sthx behavior method.
// Store Halfword Indexed
void ac_behavior( sthx )
{
    dbg_printf(" shhx r%d, r%d, r%d\n\n",rs,ra,rb);

```

```
int ea;

if(ra!=0)
    ea=GPR[ra]+GPR[rb];
else
    ea=GPR[rb];

MEM.write_half(ea,(unsigned short int)GPR[rs]);
};

//!Instruction stmw behavior method.
// Store Multiple Word
void ac_behavior( stmw )
{
    dbg_printf(" stmw r%d, %d(r%d)\n\n",rs,d,ra);

    int ea;
    unsigned int r;

    if(ra!=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    r=rs;

    while(r<=31) {
        MEM.write(ea,GPR[r]);
        r+=1;
        ea+=4;
    }
};

//!Instruction stswi behavior method.
// Store String Word Immediate
```

```
void ac_behavior( stswi )
{
    dbg_printf(" stswi r%d, r%d, %d\n\n",rs,ra,nb);

    int ea;
    unsigned int n;
    unsigned int r;
    unsigned int i,masc;

    if(ra!=0)
        ea=GPR[ra];
    else
        ea=0;

    if(nb==0)
        n=32;
    else
        n=nb;

    r=rs-1;
    i=0;

    while(n>0) {
        if(i==0)
            r=r+1;
        if(r==32)
            r=0;
        masc=mask32rlw(i,i+7);
        MEM.write_byte(ea,(unsigned char)((GPR[r] & masc) >> (24-i)));
        i=i+8;
        if(i==32)
            i=0;
        ea=ea+1;
        n=n-1;
    }
}
```

```
};

//!Instruction stswx behavior method.
// Store String Word Indexed
void ac_behavior( stswx )
{
    dbg_printf(" stswx r%d, r%d, r%d\n\n",rs,ra,rb);

    int ea;
    unsigned int n;
    unsigned int r;
    unsigned int i,masc;

    if(ra!=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];

    n=XER_TBC_read();

    r=rs-1;
    i=0;

    while(n>0) {
        if(i==0)
            r=r+1;
        if(r==32)
            r=0;
        masc=mask32rlw(i,i+7);
        MEM.write_byte(ea,(unsigned char)((GPR[r] & masc) >> (24-i)));
        i=i+8;
        if(i==32)
            i=0;
        ea=ea+1;
    }
}
```



```
        n=n-1;
    }

};

//!Instruction stw behavior method.
// Store Word
void ac_behavior( stw )
{
    dbg_printf(" stw r%d, %d(r%d)\n\n",rs,d,ra);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+(short int)d;
    else
        ea=(short int)d;

    MEM.write(ea,(unsigned int)GPR[rs]);
};

//!Instruction stwbrx behavior method.
// Store Word Byte-Reverse Indexed
void ac_behavior( stwbrx )
{
    dbg_printf(" stwbrx r%d, r%d, r%d\n\n",rs,ra,rb);

    int ea;

    if(ra!=0)
        ea=GPR[ra]+GPR[rb];
    else
        ea=GPR[rb];

    MEM.write(ea,(((GPR[rs] & 0x000000FF) << 24) |
```

```
        ((GPR[rs] & 0x0000FF00) << 8 ) |
        ((GPR[rs] & 0x00FF0000) >> 8 ) |
        ((GPR[rs] & 0xFF000000) >> 24));
};

//!Instruction stwu behavior method.
// Store Word with Update
void ac_behavior( stwu )
{
    dbg_printf(" stwu r%d, %d(r%d)\n\n",rs,d,ra);

    int ea=GPR[ra]+(short int)d;

    MEM.write(ea,(unsigned int)GPR[rs]);
    GPR[ra] = ea;
};

//!Instruction stwux behavior method.
// Store Word with Update Indexed
void ac_behavior( stwux )
{
    dbg_printf("stwux r%d, r%d, r%d\n\n",rs,ra,rb);

    int ea=GPR[ra]+GPR[rb];

    MEM.write(ea,GPR[rs]);
    GPR[ra] = ea;
};

//!Instruction stwx behavior method.
// Store Word Indexed
void ac_behavior( stwx )
{
    dbg_printf(" stwx r%d, r%d, r%d\n\n",rs,ra,rb);
```

```

int ea;

if(ra!=0)
    ea=GPR[ra]+GPR[rb];
else
    ea=GPR[rb];

MEM.write(ea,(unsigned int)GPR[rs]);
};

//!Instruction subf behavior method.
// Subtract From
void ac_behavior( subf )
{
    dbg_printf(" subf%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r
    int result=~GPR[ra] + GPR[rb] + 1;

    if (oe==1)
        add_XER_OV_SO_update(result,~GPR[ra],GPR[rb],1);

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction subfc behavior method.
// Subtract From Carrying
void ac_behavior( subfc )
{
    dbg_printf(" subfc%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r
    int result=~GPR[ra] + GPR[rb] + 1;

```

```

add_XER_CA_update(result, ~GPR[ra], GPR[rb], 1);

if (oe==1)
    add_XER_OV_SO_update(result, ~GPR[ra], GPR[rb], 1);

if (rc==1)
    CR0_update(result);

GPR[rt] = result;
};

//!Instruction subfe behavior method.
// Subtract From Extended
void ac_behavior( subfe )
{
    dbg_printf(" subfe%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),
    int result=~GPR[ra] + GPR[rb] + XER_CA_read();

    add_XER_CA_update(result, ~GPR[ra], GPR[rb], XER_CA_read());

    if (oe==1)
        add_XER_OV_SO_update(result, ~GPR[ra], GPR[rb], XER_CA_read());

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction subfic behavior method.
// Subtract From Immediate Carrying
void ac_behavior( subfic )
{

```

```

    dbg_printf(" subfic r%d, r%d, %d\n\n",rt,ra,d);
    int ime32=d;
    int result=~GPR[ra]+ime32+1;

    add_XER_CA_update(result,~GPR[ra],ime32,1);

    GPR[rt] = result;
};

//!Instruction subfme behavior method.
// Subtract from Minus One Extended
void ac_behavior( subfme )
{
    dbg_printf(" subfme%s r%d, r%d\n\n",(oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt,r

    int result=~GPR[ra]+XER_CA_read()+(-1);

    add_XER_CA_update(result,~GPR[ra],XER_CA_read(),-1);

    if (oe==1)
        add_XER_OV_SO_update(result,~GPR[ra],XER_CA_read(),-1);

    if (rc==1)
        CR0_update(result);

    GPR[rt] = result;
};

//!Instruction subfze behavior method.
// Subtract from Zero Extended
void ac_behavior( subfze )
{
    dbg_printf(" subfze%s r%d, r%d\n\n",(oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt,r

    int result=~GPR[ra]+XER_CA_read();

```

```

add_XER_CA_update(result,~GPR[ra],XER_CA_read(),0);

if (oe==1)
    add_XER_OV_SO_update(result,~GPR[ra],XER_CA_read(),0);

if (rc==1)
    CR0_update(result);

GPR[rt] = result;
};

//!Instruction xor behavior method.
// XOR
void ac_behavior( xxor )
{
    dbg_printf(" xor%s r%d, r%d, r%d\n\n", (rc==1?".":""),ra,rs,rb);
    int result=GPR[rs] ^GPR[rb];

    if (rc==1)
        CR0_update(result);

    GPR[ra] = result;

};

//!Instruction xori behavior method.
void ac_behavior( xori )
{
    dbg_printf(" xori r%d, r%d, %d\n\n",ra,rs,ui);

    GPR[ra] = GPR[rs] ^(int)((unsigned short int)ui);
};

//!Instruction xoris behavior method.

```

```

void ac_behavior( xoris )
{
    dbg_printf(" xoris r%d, r%d, %d\n\n",ra,rs,ui);

    GPR[ra] = GPR[rs] ^(((int)((unsigned short int)ui)) << 16);
};

```

5.2 Precisão de Ciclos

5.2.1 ppc405_ca.ac

```

/*****/
/* The ArchC PowerPC405 cycle-accurate model.          */
/*                                                     */
/* Author:                                             */
/* Sandro Carvalho - sandroc@inf.ufsc.br              */
/*                                                     */
/* For more information on ArchC, please visit:       */
/* http://www.archc.org                               */
/*                                                     */
/* System Design Automation Lab (LAPS)                */
/* INF-UFSC                                           */
/* http://www.laps.inf.ufsc.br                       */
/*****/

```

```

AC_ARCH(ppc405_ca) {

    ac_wordsize 32;

    ac_mem MEM:8M;

    ac_regbank GPR:32;

    ac_format F_FE_DE = "%npc:32";

```

```

    ac_format F_DE_EX = "%npc:32 %data1:32 %data2:32 %data3:32 %ra:5 %rb:5 %r3:5
%mb:5 %me:5 %sh:5 %nb:5 %u3a:3 %u3b:3 %rf:10 %regwrite:1 %oe:1 %rc:1 %ui:16
%si:16:s %d:16:s %ldwrite:1 %stwrite:1 %bo:5 %bi:5 %bd:14:s %aa:1 %lk:1 %li:24:s";
    ac_format F_EX_WB = "%alures:32 %wdata:32 %ea:32 %rdest:5 %regwrite:1 %stwrite:1
%ldwrite:1 %lddest:5";
    ac_format F_WB_LW = "%lddata:32 %lddest:5 %ldwrite:1";

/* FETch, DEcode, EXecute, Write-Back and Load Write-back */
ac_reg<F_FE_DE> FE_DE;
ac_reg<F_DE_EX> DE_EX;
ac_reg<F_EX_WB> EX_WB;
ac_reg<F_WB_LW> WB_LW;

ac_stage FE,DE,EX,WB,LW;

ac_reg SPRG4;
ac_reg SPRG5;
ac_reg SPRG6;
ac_reg SPRG7;
ac_reg USPRG0;

ac_reg MSR;

ac_reg EVPR;
ac_reg SRR0;
ac_reg SRR1;

ac_reg XER;
ac_reg CR;
ac_reg LR;
ac_reg CTR;

ARCH_CTOR(ppc405_ca) {
    ac_isa("ppc405_ca_isa.ac");
    set_endian("big");

```



```
};

};
```

5.2.2 ppc405_ca_isa.ac

```

/*****/
/* The ArchC PowerPC405 cycle-accurate model.          */
/*                                                    */
/* Author:                                             */
/* Sandro Carvalho - sandroc@inf.ufsc.br              */
/*                                                    */
/* For more information on ArchC, please visit:       */
/* http://www.archc.org                               */
/*                                                    */
/* System Design Automation Lab (LAPS)                */
/* INF-UFSC                                           */
/* http://www.laps.inf.ufsc.br                        */
/*****/

```

```
AC_ISA(ppc405_ca) {
```

```
    ac_format I1 = "%opcd:6 %li:24:s %aa:1 %lk:1";
```

```
    ac_format B1 = "%opcd:6 %bo:5 %bi:5 %bd:14:s %aa:1 %lk:1";
```

```
    ac_format SC1 = "%opcd:6 0x00:5 0x00:5 0x00:4 %lev:7 0x00:3 0x01:1 0x00:1";
```

```
    ac_format D1 = "%opcd:6 %rt:5 %ra:5 %d:16:s";
```

```
    ac_format D2 = "%opcd:6 %rs:5 %ra:5 %si:16:s";
```

```
    ac_format D3 = "%opcd:6 %rs:5 %ra:5 %d:16:s";
```

```
    ac_format D4 = "%opcd:6 %rs:5 %ra:5 %ui:16";
```

```
    ac_format D5 = "%opcd:6 %bf:3 0x00:1 %l:1 %ra:5 %si:16:s";
```

```
    ac_format D6 = "%opcd:6 %bf:3 0x00:1 %l:1 %ra:5 %ui:16";
```

```
    ac_format D7 = "%opcd:6 %to:5 %ra:5 %si:16:s";
```

```
ac_format X1 = "%opcd:6 %rt:5 %ra:5 %rb:5 %xog:10 %rc:1";
ac_format X2 = "%opcd:6 %rt:5 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X3 = "%opcd:6 %rt:5 %ra:5 %nb:5 %xog:10 0x00:1";
ac_format X4 = "%opcd:6 %rt:5 %ra:5 %ws:5 %xog:10 0x00:1";
ac_format X5 = "%opcd:6 %rt:5 0x00:5 %rb:5 %xog:10 0x00:1";
ac_format X6 = "%opcd:6 %rt:5 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X7 = "%opcd:6 %rs:5 %ra:5 %rb:5 %xog:10 %rc:1";
ac_format X8 = "%opcd:6 %rs:5 %ra:5 %rb:5 %xog:10 0x01:1";
ac_format X9 = "%opcd:6 %rs:5 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X10 = "%opcd:6 %rs:5 %ra:5 %nb:5 %xog:10 0x00:1";
ac_format X11 = "%opcd:6 %rs:5 %ra:5 %ws:5 %xog:10 0x00:1";
ac_format X12 = "%opcd:6 %rs:5 %ra:5 %sh:5 %xog:10 %rc:1";
ac_format X13 = "%opcd:6 %rs:5 %ra:5 0x00:5 %xog:10 %rc:1";
ac_format X14 = "%opcd:6 %rs:5 0x00:5 %rb:5 %xog:10 0x00:1";
ac_format X15 = "%opcd:6 %rs:5 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X16 = "%opcd:6 %bf:3 0x00:1 %l:1 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X17 = "%opcd:6 %bf:3 0x00:2 %bfa:3 0x00:2 0x00:5 %xog:10 %rc:1";
ac_format X18 = "%opcd:6 %bf:3 0x00:2 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X19 = "%opcd:6 %bf:3 0x00:2 0x00:5 %u:5 %xog:10 %rc:1";
ac_format X20 = "%opcd:6 %bf:3 0x00:2 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X21 = "%opcd:6 %to:5 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X22 = "%opcd:6 %bt:5 0x00:5 0x00:5 %xog:10 %rc:1";
ac_format X23 = "%opcd:6 0x00:5 %ra:5 %rb:5 %xog:10 0x00:1";
ac_format X24 = "%opcd:6 0x00:5 0x00:5 0x00:5 %xog:10 0x00:1";
ac_format X25 = "%opcd:6 0x00:5 0x00:5 %e:1 0x00:4 %xog:10 0x00:1";

ac_format XL1 = "%opcd:6 %bt:5 %ba:5 %bb:5 %xog:10 0x00:1";
ac_format XL2 = "%opcd:6 %bo:5 %bi:5 0x00:3 %bh:2 %xog:10 %lk:1";
ac_format XL3 = "%opcd:6 %bf:3 0x00:2 %bfa:3 0x00:2 0x00:5 %xog:10 0x00:1";
ac_format XL4 = "%opcd:6 0x00:5 0x00:5 0x00:5 %xog:10 0x00:1";

ac_format XFX1 = "%opcd:6 %rt:5 %sprf:10 %xog:10 0x00:1";
ac_format XFX2 = "%opcd:6 %rt:5 %dcrf:10 %xog:10 0x00:1";
ac_format XFX3 = "%opcd:6 %rs:5 0x00:1 %xfm:8 0x00:1 %xog:10 0x00:1";
```

```
ac_format XFX4 = "%opcd:6 %rs:5 %sprf:10 %xog:10 0x00:1";
ac_format XFX5 = "%opcd:6 %rs:5 %dcrf:10 %xog:10 0x00:1";

ac_format X01 = "%opcd:6 %rt:5 %ra:5 %rb:5 %oe:1 %xos:9 %rc:1";
ac_format X02 = "%opcd:6 %rt:5 %ra:5 %rb:5 0x00:1 %xos:9 %rc:1";
ac_format X03 = "%opcd:6 %rt:5 %ra:5 0x00:5 %oe:1 %xos:9 %rc:1";

ac_format M1 = "%opcd:6 %rs:5 %ra:5 %rb:5 %mb:5 %me:5 %rc:1";
ac_format M2 = "%opcd:6 %rs:5 %ra:5 %sh:5 %mb:5 %me:5 %rc:1";

ac_instr<I1> b;

ac_instr<B1> bc;

ac_instr<SC1> sc;

ac_instr<D1> addi, addic, addic_, addis, lbz, lbzu, lha, lhau, lhz,
    lhzu, lmw, lwz, lwzu, mulli, subfic;

ac_instr<D3> stb, stbu, sth, sthu, stmw, stw, stwu;

ac_instr<D4> andi_, andis_, ori, oris, xori, xoris;

ac_instr<D5> cmpi;

ac_instr<D6> cmpli;

ac_instr<X1> mulchw, mulchwu, mulhhw, mulhhwu,
    mullhw, mullhwu;

ac_instr<X2> lbzux, lbzx, lhaux, lhax, lhbrx, lhzux, lhzx, lswx,
    lwbrx, lwzux, lwzx;

ac_instr<X3> lswi;
```

```
ac_instr<X6> mfcr;
```

```
ac_instr<X7> ande, andc, eqv, nand,  
            nor, ore, orc, slw, sraw,  
            srw, xxor;
```

```
ac_instr<X9> stbux, stbx, sthbrx, sthux, stswx, stwbrx, stwux, stwx,  
            sthx;
```

```
ac_instr<X10> stswi;
```

```
ac_instr<X12> srawi;
```

```
ac_instr<X13> cntlzw, extsb, extsh;
```

```
ac_instr<X16> cmp, cmpl;
```

```
ac_instr<X18> mcrxr;
```

```
ac_instr<XL1> crand, crandc, creqv, crnand, crnor, cror, crorc, crxor;
```

```
ac_instr<XL2> bcctr, bclr;
```

```
ac_instr<XL3> mcrf;
```

```
ac_instr<XFX1> mfspr;
```

```
ac_instr<XFX3> mtcrf;
```

```
ac_instr<XFX4> mtspr;
```

```
ac_instr<X01> add, addc, adde, mullw, divw, divwu,  
            subf, subfc, subfe;
```

```
ac_instr<X01> macchw, macchws, macchwsu, macchwu,
```

```

machhw, machhws, machhwsu, machhwu,
maclhw, maclhws, maclhwsu, maclhwu,
nmacchw, nmacchws, nmachhw, nmachhws,
nmaclhw, nmaclhws;

```

```
ac_instr<X02> mulhw, mulhwu;
```

```
ac_instr<X03> addme, addze, neg, subfme, subfze;
```

```
ac_instr<M1> rlwnm;
```

```
ac_instr<M2> rlwimi, rlwinm;
```

```
ac_asm_map reg {
    ""[0..31] = [0..31];
}

```

```
ISA_CTOR(ppc405) {
```

```

add.set_asm("add  %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
add.set_asm("add. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
add.set_asm("addo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
add.set_asm("addo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
add.set_decoder(opcd=31, xos=266);

addc.set_asm("addc  %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
addc.set_asm("addc. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
addc.set_asm("addco %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
addc.set_asm("addco. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
addc.set_decoder(opcd=31, xos=10);

adde.set_asm("adde  %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
adde.set_asm("adde. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
adde.set_asm("addeo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
adde.set_asm("addeo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);

```

```
adde.set_decoder(opcd=31, xos=138);

addi.set_asm("li %reg, %exp", rt, ra=0, d);
addi.set_asm("li %reg, %expHc@ha", rt, ra=0, d);
addi.set_asm("la %reg, %exp@l(%imm)", rt, d, ra );
addi.set_asm("addi %reg, %reg, %exp", rt, ra, d);
addi.set_decoder(opcd=14);

addic.set_asm("addic %reg, %reg, %exp", rt, ra, d);
addic.set_decoder(opcd=12);

addic_.set_asm("addic. %reg, %reg, %exp", rt, ra, d);
addic_.set_decoder(opcd=13);

addis.set_asm("lis %reg, %exp", rt, ra=0, d);
addis.set_asm("lis %reg, %expHc@ha", rt, ra=0, d);
addis.set_asm("addis %reg, %reg, %exp", rt, ra, d);
addis.set_decoder(opcd=15);

addme.set_asm("addme %reg, %reg", rt, ra, oe=0, rc=0);
addme.set_asm("addme. %reg, %reg", rt, ra, oe=0, rc=1);
addme.set_asm("addmeo %reg, %reg", rt, ra, oe=1, rc=0);
addme.set_asm("addmeo. %reg, %reg", rt, ra, oe=1, rc=1);
addme.set_decoder(opcd=31, xos=234);

addze.set_asm("addze %reg, %reg", rt, ra, oe=0, rc=0);
addze.set_asm("addze. %reg, %reg", rt, ra, oe=0, rc=1);
addze.set_asm("addzeo %reg, %reg", rt, ra, oe=1, rc=0);
addze.set_asm("addzeo. %reg, %reg", rt, ra, oe=1, rc=1);
addze.set_decoder(opcd=31, xos=202);

ande.set_asm("and %reg, %reg, %reg", ra, rs, rb, rc=0);
ande.set_asm("and. %reg, %reg, %reg", ra, rs, rb, rc=1);
ande.set_decoder(opcd=31, xog=28);
```

```
andc.set_asm("andc %reg, %reg, %reg", ra, rs, rb, rc=0);
andc.set_asm("andc. %reg, %reg, %reg", ra, rs, rb, rc=1);
andc.set_decoder(opcd=31, xog=60);
```

```
andi_.set_asm("andi. %reg, %reg, %imm", ra, rs, ui);
andi_.set_decoder(opcd=28);
```

```
andis_.set_asm("andis. %reg, %reg, %imm", ra, rs, ui);
andis_.set_decoder(opcd=29);
```

```
b.set_asm("b %addrRAu", li, aa=0, lk=0);
b.set_asm("ba %addrRAu", li, aa=1, lk=0);
b.set_asm("bl %addrRAu", li, aa=0, lk=1);
b.set_asm("bla %addrRAu", li, aa=1, lk=1);
b.set_decoder(opcd=18);
```

```
bc.set_asm("bc %imm, %exp, %addrRAu", bo, bi, bd, aa=0, lk=0);
bc.set_asm("bca %imm, %exp, %addrRAu", bo, bi, bd, aa=1, lk=0);
bc.set_asm("bcl %imm, %exp, %addrRAu", bo, bi, bd, aa=0, lk=1);
bc.set_asm("bcla %imm, %exp, %addrRAu", bo, bi, bd, aa=1, lk=1);
bc.set_decoder(opcd=16);
```

```
bcctr.set_asm("bcctr", bo=0x14, bi=0, bh=0, lk=0);
bcctr.set_asm("bcctl", bo=0x14, bi=0, bh=0, lk=1);
bcctr.set_asm("bcctr %reg, %reg, %reg", bo, bi, bh, lk=0);
bcctr.set_asm("bcctl %reg, %reg, %reg", bo, bi, bh, lk=1);
bcctr.set_decoder(opcd=19, xog=528);
```

```
bclr.set_asm("bclr", bo=0x14, bi=0, bh=0, lk=0);
bclr.set_asm("bclr %reg, %reg, %reg", bo, bi, bh, lk=0);
bclr.set_asm("bclrl %reg, %reg, %reg", bo, bi, bh, lk=1);
bclr.set_decoder(opcd=19, xog=16);
```

```
cmp.set_asm("cmpw %imm, %imm, %imm", bf, ra, rb, l=0);
cmp.set_asm("cmp %imm, %imm, %reg, %reg", bf, l, ra, rb);
```

```
cmp.set_decoder(opcd=31, l=0, xog=0);

cmpi.set_asm("cmpwi %imm, %reg, %imm", bf, ra, si, l=0);
cmpi.set_asm("cmpi %reg, %imm, %reg, %imm", bf, l, ra, si);
cmpi.set_decoder(opcd=11, l=0);

cmpl.set_asm("cmplw %reg, %reg, %reg", bf, ra, rb, l=0);
cmpl.set_asm("cmpl %imm, %imm, %reg, %reg", bf, l, ra, rb);
cmpl.set_decoder(opcd=31, l=0, xog=32);

cmpli.set_asm("cmplwi %reg, %reg, %imm", bf, ra, ui, l=0);
cmpli.set_asm("cmpli %reg, %imm, %reg, %imm", bf, l, ra, ui);
cmpli.set_decoder(opcd=10, l=0);

cntlzw.set_asm("cntlzw %reg, %reg", ra, rs, rc=0);
cntlzw.set_asm("cntlzw. %reg, %reg", ra, rs, rc=1);
cntlzw.set_decoder(opcd=31, xog=26);

crand.set_asm("crand %reg, %reg, %reg", bt, ba, bb);
crand.set_decoder(opcd=19, xog=257);

crandc.set_asm("crandc %reg, %reg, %reg", bt, ba, bb);
crandc.set_decoder(opcd=19, xog=129);

creqv.set_asm("creqv %reg, %reg, %reg", bt, ba, bb);
creqv.set_decoder(opcd=19, xog=289);

crnand.set_asm("crnand %reg, %reg, %reg", bt, ba, bb);
crnand.set_decoder(opcd=19, xog=225);

crnor.set_asm("crnor %reg, %reg, %reg", bt, ba, bb);
crnor.set_decoder(opcd=19, xog=33);

cror.set_asm("cror %reg, %reg, %reg", bt, ba, bb);
cror.set_decoder(opcd=19, xog=449);
```



```
crorc.set_asm("crorc %reg, %reg, %reg", bt, ba, bb);  
crorc.set_decoder(opcd=19, xog=417);
```

```
crxor.set_asm("crxor %reg, %reg, %reg", bt, ba, bb);  
crxor.set_decoder(opcd=19, xog=193);
```

```
divw.set_asm("divw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);  
divw.set_asm("divw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);  
divw.set_asm("divwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);  
divw.set_asm("divwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);  
divw.set_decoder(opcd=31, xos=491);
```

```
divwu.set_asm("divwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);  
divwu.set_asm("divwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);  
divwu.set_asm("divwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);  
divwu.set_asm("divwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);  
divwu.set_decoder(opcd=31, xos=459);
```

```
eqv.set_asm("eqv %reg, %reg, %reg", ra, rs, rb, rc=0);  
eqv.set_asm("eqv. %reg, %reg, %reg", ra, rs, rb, rc=1);  
eqv.set_decoder(opcd=31, xog=284);
```

```
extsb.set_asm("extsb %reg, %reg", ra, rs, rc=0);  
extsb.set_asm("extsb. %reg, %reg", ra, rs, rc=1);  
extsb.set_decoder(opcd=31, xog=954);
```

```
extsh.set_asm("extsh %reg, %reg", ra, rs, rc=0);  
extsh.set_asm("extsh. %reg, %reg", ra, rs, rc=1);  
extsh.set_decoder(opcd=31, xog=922);
```

```
lbz.set_asm("lbz %reg, %imm(%reg)", rt, d, ra);  
lbz.set_asm("lbz %reg, %exp@l(%reg)", rt, d, ra);  
lbz.set_decoder(opcd=34);
```

```
lbzu.set_asm("lbzu %reg, %imm(%reg)", rt, d, ra);
lbzu.set_asm("lbzu %reg, %exp@l(%reg)", rt, d, ra);
lbzu.set_decoder(opcd=35);

lbzux.set_asm("lbzux %reg, %reg, %reg", rt, ra, rb);
lbzux.set_decoder(opcd=31, xog=119);

lbzx.set_asm("lbzx %reg, %reg, %reg", rt, ra, rb);
lbzx.set_decoder(opcd=31, xog=87);

lha.set_asm("lha %reg, %imm(%reg)", rt, d, ra);
lha.set_asm("lha %reg, %exp@l(%reg)", rt, d, ra);
lha.set_decoder(opcd=42);

lhau.set_asm("lhau %reg, %imm(%reg)", rt, d, ra);
lhau.set_asm("lhau %reg, %exp@l(%reg)", rt, d, ra);
lhau.set_decoder(opcd=43);

lhaux.set_asm("lhaux %reg, %reg, %reg", rt, ra, rb);
lhaux.set_decoder(opcd=31, xog=375);

lhax.set_asm("lhax %reg, %reg, %reg", rt, ra, rb);
lhax.set_decoder(opcd=31, xog=343);

lhbrx.set_asm("lhbrx %reg, %reg, %reg", rt, ra, rb);
lhbrx.set_decoder(opcd=31, xog=790);

lhz.set_asm("lhz %reg, %imm(%reg)", rt, d, ra);
lhz.set_asm("lhz %reg, %exp@l(%reg)", rt, d, ra);
lhz.set_decoder(opcd=40);

lhzu.set_asm("lhzu %reg, %imm(%reg)", rt, d, ra);
lhzu.set_asm("lhzu %reg, %exp@l(%reg)", rt, d, ra);
lhzu.set_decoder(opcd=41);
```

```
lhzux.set_asm("lhzux %reg, %reg, %reg", rt, ra, rb);
lhzux.set_decoder(opcd=31, xog=311);

lhzx.set_asm("lhzx %reg, %reg, %reg", rt, ra, rb);
lhzx.set_decoder(opcd=31, xog=279);

lmw.set_asm("lmw %reg, %imm(%reg)", rt, d, ra);
lmw.set_asm("lmw %reg, %exp@l(%reg)", rt, d, ra);
lmw.set_decoder(opcd=46);

lswi.set_asm("lswi %imm, %imm, %imm", rt, ra, nb);
lswi.set_decoder(opcd=31, xog=597);

lswx.set_asm("lswx %reg, %reg, %reg", rt, ra, rb);
lswx.set_decoder(opcd=31, xog=533);

lwbrx.set_asm("lwbrx %reg, %reg, %reg", rt, ra, rb);
lwbrx.set_decoder(opcd=31, xog=534);

lwz.set_asm("lwz %imm, %imm(%imm)", rt, d, ra);
lwz.set_asm("lwz %imm, %exp@l(%imm)", rt, d, ra);
lwz.set_decoder(opcd=32);

lwzu.set_asm("lwzu %reg, %imm(%reg)", rt, d, ra);
lwzu.set_asm("lwzu %reg, %exp@l(%reg)", rt, d, ra);
lwzu.set_decoder(opcd=33);

lwzux.set_asm("lwzux %reg, %reg, %reg", rt, ra, rb);
lwzux.set_decoder(opcd=31, xog=55);

lwzx.set_asm("lwzx %reg, %reg, %reg", rt, ra, rb);
lwzx.set_decoder(opcd=31, xog=23);

macchw.set_asm("macchw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchw.set_asm("macchw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
```

```
macchw.set_asm("macchwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
macchw.set_asm("macchwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchw.set_decoder(opcd=4, xos=172);

macchws.set_asm("macchws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchws.set_asm("macchws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
macchws.set_asm("macchwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
macchws.set_asm("macchwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchws.set_decoder(opcd=4, xos=236);

macchwsu.set_asm("macchwsu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchwsu.set_asm("macchwsu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
macchwsu.set_asm("macchwsuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
macchwsu.set_asm("macchwsuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchwsu.set_decoder(opcd=4, xos=204);

macchwu.set_asm("macchwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
macchwu.set_asm("macchwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
macchwu.set_asm("macchwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
macchwu.set_asm("macchwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
macchwu.set_decoder(opcd=4, xos=140);

machhw.set_asm("machhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
machhw.set_asm("machhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhw.set_asm("machhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhw.set_asm("machhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhw.set_decoder(opcd=4, xos=44);

machhws.set_asm("machhws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
machhws.set_asm("machhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhws.set_asm("machhwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhws.set_asm("machhwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhws.set_decoder(opcd=4, xos=108);

machhwsu.set_asm("machhwsu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
```

```
machhwsu.set_asm("machhwsu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhwsu.set_asm("machhwsuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhwsu.set_asm("machhwsuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhwsu.set_decoder(opcd=4, xos=76);
```

```
machhwu.set_asm("machhwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
machhwu.set_asm("machhwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
machhwu.set_asm("machhwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
machhwu.set_asm("machhwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
machhwu.set_decoder(opcd=4, xos=12);
```

```
maclhw.set_asm("maclhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhw.set_asm("maclhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhw.set_asm("maclhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhw.set_asm("maclhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhw.set_decoder(opcd=4, xos=428);
```

```
maclhws.set_asm("maclhws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhws.set_asm("maclhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhws.set_asm("maclhwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhws.set_asm("maclhwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhws.set_decoder(opcd=4, xos=492);
```

```
maclhwsu.set_asm("maclhwsu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhwsu.set_asm("maclhwsu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhwsu.set_asm("maclhwsuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhwsu.set_asm("maclhwsuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhwsu.set_decoder(opcd=4, xos=460);
```

```
maclhwu.set_asm("maclhwu %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
maclhwu.set_asm("maclhwu. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
maclhwu.set_asm("maclhwuo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
maclhwu.set_asm("maclhwuo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
maclhwu.set_decoder(opcd=4, xos=396);
```

```
mcrf.set_asm("mcrf %imm, %imm", bf, bfa);
mcrf.set_decoder(opcd=19, xog=0);

mcrxr.set_asm("mcrxr %imm", bf);
mcrxr.set_decoder(opcd=31, xog=512);

mfcr.set_asm("mfcr %imm", rt);
mfcr.set_decoder(opcd=31, xog=19);

/* INCOMPLETE IMPLEMENTATION! */
mfspr.set_asm("mfctr %imm", rt, sprf=0x120);
mfspr.set_asm("mflr %imm", rt, sprf=0x100);
mfspr.set_asm("mfspr %imm, %imm", rt, sprf);
mfspr.set_decoder(opcd=31, xog=339);

mtcrf.set_asm("mtcrf %imm, %imm", xfm, rs);
mtcrf.set_decoder(opcd=31, xog=144);

/* INCOMPLETE IMPLEMENTATION! */
mtspr.set_asm("mtctr %imm", rs, sprf=0x120);
mtspr.set_asm("mtlr %imm", rs, sprf=0x100);
mtspr.set_asm("mtspr %imm, %imm", sprf, rs);
mtspr.set_decoder(opcd=31, xog=467);

mulchw.set_asm("mulchw %reg, %reg, %reg", rt, ra, rb, rc=0);
mulchw.set_asm("mulchw. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulchw.set_decoder(opcd=4, xog=168);

mulchwu.set_asm("mulchwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mulchwu.set_asm("mulchwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulchwu.set_decoder(opcd=4, xog=136);

mulhhw.set_asm("mulhhw %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhhw.set_asm("mulhhw. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulhhw.set_decoder(opcd=4, xog=40);
```

```
mulhhwu.set_asm("mulhhwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhhwu.set_asm("mulhhwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulhhwu.set_decoder(opcd=4, xog=8);
```

```
mulhw.set_asm("mulhw %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhw.set_asm("mulhw. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulhw.set_decoder(opcd=31, xos=75);
```

```
mulhwu.set_asm("mulhwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mulhwu.set_asm("mulhwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mulhwu.set_decoder(opcd=31, xos=11);
```

```
mullhw.set_asm("mullhw %reg, %reg, %reg", rt, ra, rb, rc=0);
mullhw.set_asm("mullhw. %reg, %reg, %reg", rt, ra, rb, rc=1);
mullhw.set_decoder(opcd=4, xog=424);
```

```
mullhwu.set_asm("mullhwu %reg, %reg, %reg", rt, ra, rb, rc=0);
mullhwu.set_asm("mullhwu. %reg, %reg, %reg", rt, ra, rb, rc=1);
mullhwu.set_decoder(opcd=4, xog=392);
```

```
mulli.set_asm("mulli %reg, %reg, %exp", rt, ra, d);
mulli.set_decoder(opcd=7);
```

```
mullw.set_asm("mullw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
mullw.set_asm("mullw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
mullw.set_asm("mullwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
mullw.set_asm("mullwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
mullw.set_decoder(opcd=31, xos=235);
```

```
nand.set_asm("nand %reg, %reg, %reg", ra, rs, rb, rc=0);
nand.set_asm("nand. %reg, %reg, %reg", ra, rs, rb, rc=1);
nand.set_decoder(opcd=31, xog=476);
```

```
neg.set_asm("neg %reg, %reg", rt, ra, oe=0, rc=0);
```

```
neg.set_asm("neg. %reg, %reg", rt, ra, oe=0, rc=1);
neg.set_asm("nego %reg, %reg", rt, ra, oe=1, rc=0);
neg.set_asm("nego. %reg, %reg", rt, ra, oe=1, rc=1);
neg.set_decoder(opcd=31, xos=104);

nmacchw.set_asm("nmacchw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmacchw.set_asm("nmacchw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmacchw.set_asm("nmacchw. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmacchw.set_asm("nmacchw. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmacchw.set_decoder(opcd=4, xos=174);

nmacchws.set_asm("nmacchws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmacchws.set_asm("nmacchws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmacchws.set_asm("nmacchwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmacchws.set_asm("nmacchwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmacchws.set_decoder(opcd=4, xos=238);

nmachhw.set_asm("nmachhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmachhw.set_asm("nmachhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmachhw.set_asm("nmachhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmachhw.set_asm("nmachhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmachhw.set_decoder(opcd=4, xos=46);

nmachhws.set_asm("nmachhws %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmachhws.set_asm("nmachhws. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmachhws.set_asm("nmachhwso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmachhws.set_asm("nmachhwso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmachhws.set_decoder(opcd=4, xos=110);

nmaclhw.set_asm("nmaclhw %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmaclhw.set_asm("nmaclhw. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmaclhw.set_asm("nmaclhwo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmaclhw.set_asm("nmaclhwo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmaclhw.set_decoder(opcd=4, xos=430);
```



```
nmaclhs.set_asm("nmaclhs %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
nmaclhs.set_asm("nmaclhs. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
nmaclhs.set_asm("nmaclhsso %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
nmaclhs.set_asm("nmaclhsso. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
nmaclhs.set_decoder(opcd=4, xos=494);

nor.set_asm("nor %reg, %reg, %reg", ra, rs, rb, rc=0);
nor.set_asm("nor. %reg, %reg, %reg", ra, rs, rb, rc=1);
nor.set_decoder(opcd=31, xog=124);

ore.set_asm("or %reg, %reg, %reg", ra, rs, rb, rc=0);
ore.set_asm("or. %reg, %reg, %reg", ra, rs, rb, rc=1);
ore.set_decoder(opcd=31, xog=444);

orc.set_asm("orc %reg, %reg, %reg", ra, rs, rb, rc=0);
orc.set_asm("orc. %reg, %reg, %reg", ra, rs, rb, rc=1);
orc.set_decoder(opcd=31, xog=412);

ori.set_asm("ori %reg, %reg, %imm", ra, rs, ui);
ori.set_decoder(opcd=24);

oris.set_asm("oris %reg, %reg, %imm", ra, rs, ui);
oris.set_decoder(opcd=25);

rlwimi.set_asm("rlwimi %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=0);
rlwimi.set_asm("rlwimi. %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=1);
rlwimi.set_decoder(opcd=20);

rlwinm.set_asm("rlwinm %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=0);
rlwinm.set_asm("rlwinm. %reg, %reg, %exp, %imm, %exp", ra, rs, sh, mb,
me, rc=1);
rlwinm.set_decoder(opcd=21);
```

```
    rlwnm.set_asm("rlwnm %imm, %imm, %imm, %imm, %imm", ra, rs, rb, mb,
me, rc=0);
    rlwnm.set_asm("rlwnm. %imm, %imm, %imm, %imm, %imm", ra, rs, rb, mb,
me, rc=1);
    rlwnm.set_decoder(opcd=23);

    sc.set_asm("sc %imm", lev);
    sc.set_decoder(opcd=17);

    slw.set_asm("slw %reg, %reg, %reg", ra, rs, rb, rc=0);
    slw.set_asm("slw. %reg, %reg, %reg", ra, rs, rb, rc=1);
    slw.set_decoder(opcd=31, xog=24);

    sraw.set_asm("sraw %reg, %reg, %reg", ra, rs, rb, rc=0);
    sraw.set_asm("sraw. %reg, %reg, %reg", ra, rs, rb, rc=1);
    sraw.set_decoder(opcd=31, xog=792);

    srawi.set_asm("srawi %reg, %reg, %reg", ra, rs, sh, rc=0);
    srawi.set_asm("srawi. %reg, %reg, %reg", ra, rs, sh, rc=1);
    srawi.set_decoder(opcd=31, xog=824);

    srw.set_asm("srw %reg, %reg, %reg", ra, rs, rb, rc=0);
    srw.set_asm("srw. %reg, %reg, %reg", ra, rs, rb, rc=1);
    srw.set_decoder(opcd=31, xog=536);

    stb.set_asm("stb %reg, %imm(%reg)", rs, d, ra);
    stb.set_asm("stb %reg, %exp@l(%reg)", rs, d, ra);
    stb.set_decoder(opcd=38);

    stbu.set_asm("stbu %reg, %imm(%reg)", rs, d, ra);
    stbu.set_asm("stbu %reg, %exp@l(%reg)", rs, d, ra);
    stbu.set_decoder(opcd=39);

    stbux.set_asm("stbux %reg, %reg, %reg", rs, ra, rb);
```

```
stbux.set_decoder(opcd=31, xog=247);

stbx.set_asm("stbx %reg, %reg, %reg", rs, ra, rb);
stbx.set_decoder(opcd=31, xog=215);

sth.set_asm("sth %reg, %imm(%reg)", rs, d, ra);
sth.set_asm("sth %reg, %exp@1(%reg)", rs, d, ra);
sth.set_decoder(opcd=44);

sthbrx.set_asm("sthbrx %reg, %reg, %reg", rs, ra, rb);
sthbrx.set_decoder(opcd=31, xog=918);

sthu.set_asm("sthu %reg, %imm(%reg)", rs, d, ra);
sthu.set_asm("sthu %reg, %exp@1(%reg)", rs, d, ra);
sthu.set_decoder(opcd=45);

sthux.set_asm("sthux %reg, %reg, %reg", rs, ra, rb);
sthux.set_decoder(opcd=31, xog=439);

sthx.set_asm("sthx %reg, %reg, %reg", rs, ra, rb);
sthx.set_decoder(opcd=31, xog=407);

stmw.set_asm("stmw %reg, %imm(%reg)", rs, d, ra);
stmw.set_asm("stmw %reg, %exp@1(%reg)", rs, d, ra);
stmw.set_decoder(opcd=47);

stswi.set_asm("stswi %reg, %reg, %reg", rs, ra, nb);
stswi.set_decoder(opcd=31, xog=725);

stswx.set_asm("stswx %reg, %reg, %reg", rs, ra, rb);
stswx.set_decoder(opcd=31, xog=661);

stw.set_asm("stw %reg, %imm(%reg)", rs, d, ra);
stw.set_asm("stw %reg, %exp@1(%reg)", rs, d, ra);
stw.set_decoder(opcd=36);
```

```
stwbrx.set_asm("stwbrx %reg, %reg, %reg", rs, ra, rb);  
stwbrx.set_decoder(opcd=31, xog=662);
```

```
stwu.set_asm("stwu %reg, %imm(%reg)", rs, d, ra);  
stwu.set_asm("stwu %reg, %imm(%reg)", rs, d, ra);  
stwu.set_decoder(opcd=37);
```

```
stwux.set_asm("stwux %reg, %reg, %reg", rs, ra, rb);  
stwux.set_decoder(opcd=31, xog=183);
```

```
stwx.set_asm("stwx %reg, %reg, %reg", rs, ra, rb);  
stwx.set_decoder(opcd=31, xog=151);
```

```
subf.set_asm("subf %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);  
subf.set_asm("subf. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);  
subf.set_asm("subfo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);  
subf.set_asm("subfo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);  
subf.set_decoder(opcd=31, xos=40);
```

```
subfc.set_asm("subfc %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);  
subfc.set_asm("subfc. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);  
subfc.set_asm("subfco %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);  
subfc.set_asm("subfco. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);  
subfc.set_decoder(opcd=31, xos=8);
```

```
subfe.set_asm("subfe %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);  
subfe.set_asm("subfe. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);  
subfe.set_asm("subfeo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);  
subfe.set_asm("subfeo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);  
subfe.set_decoder(opcd=31, xos=136);
```

```
subfic.set_asm("subfic %reg, %reg, %exp", rt, ra, d);  
subfic.set_decoder(opcd=8);
```

```

subfme.set_asm("subfme  %reg, %reg", rt, ra, oe=0, rc=0);
subfme.set_asm("subfme. %reg, %reg", rt, ra, oe=0, rc=1);
subfme.set_asm("subfmeo %reg, %reg", rt, ra, oe=1, rc=0);
subfme.set_asm("subfmeo. %reg, %reg", rt, ra, oe=1, rc=1);
subfme.set_decoder(opcd=31, xos=232);

```

```

subfze.set_asm("subfze  %reg, %reg", rt, ra, oe=0, rc=0);
subfze.set_asm("subfze. %reg, %reg", rt, ra, oe=0, rc=1);
subfze.set_asm("subfzeo %reg, %reg", rt, ra, oe=1, rc=0);
subfze.set_asm("subfzeo. %reg, %reg", rt, ra, oe=1, rc=1);
subfze.set_decoder(opcd=31, xos=200);

```

```

xxor.set_asm("xor  %reg, %reg, %reg", ra, rs, rb, rc=0);
xxor.set_asm("xor. %reg, %reg, %reg", ra, rs, rb, rc=1);
xxor.set_decoder(opcd=31, xog=316);

```

```

xori.set_asm("xori %reg, %reg, %imm", ra, rs, ui);
xori.set_decoder(opcd=26);

```

```

xoris.set_asm("xoris %reg, %reg, %imm", ra, rs, ui);
xoris.set_decoder(opcd=27);

```

```

/*****
/* Synthetic Instructions          */
/*****

```

```

pseudo_instr("mr %reg, %reg") {
    "or %0, %1, %1";
}

```

```

pseudo_instr("subi %reg, %reg, %imm") {
    "addi %0, %1, -%2";
}

```

```
pseudo_instr("subic %reg, %reg, %imm") {  
    "addic %0, %1, -%2";  
}
```

```
pseudo_instr("subic. %reg, %reg, %imm") {  
    "addic. %0, %1, -%2";  
}
```

```
pseudo_instr("subis %reg, %reg, %imm") {  
    "addis %0, %1, -%2";  
}
```

```
pseudo_instr("srwi %reg, %reg, %imm") {  
    "rlwinm %0, %1, 32-%2, %2, 31";  
}
```

```
pseudo_instr("crnot %reg, %reg") {  
    "crnor %0, %1, %1";  
}
```

```
pseudo_instr("insrwi %reg, %reg, %imm, %imm") {  
    "rlwimi %0, %1, 32-%3-%2, %3, %3+%2-1";  
}
```

```
pseudo_instr("slwi %reg, %reg, %imm") {  
    "rlwinm %0, %1, %2, 0, 31-%2";  
}
```

```
pseudo_instr("ble %imm, %addr") {  
    "bc 0x4, %0*4+1, %1";  
}
```

```
pseudo_instr("bne %imm, %addr") {  
    "bc 0x4, %0*4+2, %1";  
}
```

```

    }

    pseudo_instr("bgt %imm, %addr") {
        "bc 0xC, %0*4+1, %1";
    }

    pseudo_instr("blt %imm, %addr") {
        "bc 0xC, %0*4, %1";
    }

    pseudo_instr("bge %imm, %addr") {
        "bc 0x4, %0*4, %1";
    }

    pseudo_instr("beq %imm, %addr") {
        "bc 0xC, %0*4+2, %1";
    }

};

};

```

5.2.3 ppc405_ca_syscall.cpp

```

/*****/
/* The ArchC PowerPC405 cycle-accurate model. */
/* */
/* Author: */
/* Sandro Carvalho - sandroc@inf.ufsc.br */
/* */
/* For more information on ArchC, please visit: */
/* http://www.archc.org */
/* */
/* System Design Automation Lab (LAPS) */
/* INF-UFSC */

```

```
/* http://www.laps.inf.ufsc.br */
/*****/

#include "ppc405_ca_syscall.H"
#include "ac_resources.H"

void ppc405_ca_syscall::get_buffer(int argn, unsigned char* buf, unsigned int
size)
{
    unsigned int addr = GPR.read(3+argn);

    for (unsigned int i = 0; i<size; i++, addr++) {
        buf[i] = MEM.read_byte(addr);
    }
}

void ppc405_ca_syscall::set_buffer(int argn, unsigned char* buf, unsigned int
size)
{
    unsigned int addr = GPR.read(3+argn);

    for (unsigned int i = 0; i<size; i++, addr++) {
        MEM.write_byte(addr, buf[i]);
    }
}

void ppc405_ca_syscall::set_buffer_noinvert(int argn, unsigned char* buf, unsigned
int size)
{
    unsigned int addr = GPR.read(3+argn);

    for (unsigned int i = 0; i<size; i+=4, addr+=4) {
        MEM.write(addr, *(unsigned int *) &buf[i]);
    }
}
```



```
int ppc405_ca_syscall::get_int(int argn)
{
    return GPR.read(3+argn);
}

void ppc405_ca_syscall::set_int(int argn, int val)
{
    GPR.write(3+argn, val);
}

void ppc405_ca_syscall::return_from_syscall()
{
    unsigned int oldr1;
    unsigned int oldr31;
    oldr1=MEM.read(GPR.read(1));
    oldr31=MEM.read(GPR.read(1)+28);
    GPR.write(1,oldr1);
    GPR.write(31,oldr31);
    ac_resources::ac_pc=LR.read();
}

void ppc405_ca_syscall::set_prog_args(int argc, char **argv)
{
    extern unsigned AC_RAM_END;
    int i, j, base;

    unsigned int ac_argv[30];
    char ac_argstr[512];

    base = AC_RAM_END - 512;
    for (i=0, j=0; i<argc; i++) {
        int len = strlen(argv[i]) + 1;
        ac_argv[i] = base + j;
        memcpy(&ac_argstr[j], argv[i], len);
    }
}
```

```

    j += len;
}

//Write argument string
GPR.write(3, AC_RAM_END-512);
set_buffer(0, (unsigned char*) ac_argstr, 512);

//Write string pointers
GPR.write(3, AC_RAM_END-512-120);
set_buffer_noinvert(0, (unsigned char*) ac_argv, 120);

//Set r3 to the argument count
GPR.write(3, argc);

//Set r4 to the string pointers
GPR.write(4, AC_RAM_END-512-120);
}

```

5.2.4 ppc405_ca-isa.cpp

```

/*****/
/* The ArchC PowerPC405 cycle-accurate model. */
/* Instruction Set Architecture Implementation */
/* Author: */
/* Sandro Carvalho - sandroc@inf.ufsc.br */
/* */
/* For more information on ArchC, please visit: */
/* http://www.archc.org */
/* */
/* System Design Automation Lab (LAPS) */
/* INF-UFSC */
/* http://www.laps.inf.ufsc.br */
/*****/

#include "ppc405_ca-isa.H"

```

```
#include "ac_isa_init.cpp"

//If you want debug information for this model, uncomment next line
//#define DEBUG_MODEL
#include "ac_debug_model.H"

// Forwarded values
ac_Uword ex_value1, ex_value2, ex_value3;

//Compute CRO fields LT, GT, EQ, SO
//XER.SO must be updated by instruction before the use of this routine!
//Arguments:
//int result -> The result register
inline void CR0_update(unsigned int result) {

    /* LT field */
    if((result & 0x80000000) >> 31)
        CR.write(CR.read() | 0x80000000); /* 1 */
    else
        CR.write(CR.read() & 0x7FFFFFFF); /* 0 */

    /* GT field */
    if(((~result & 0x80000000) >> 31) && (result!=0))
        CR.write(CR.read() | 0x40000000); /* 1 */
    else
        CR.write(CR.read() & 0xBFFFFFFF); /* 0 */

    /* EQ field */
    if(result==0)
        CR.write(CR.read() | 0x20000000); /* 1 */
    else
        CR.write(CR.read() & 0xDFFFFFFF); /* 0 */

    /* SO field */
    if(XER.read() & 0x80000000)
```

```

        CR.write(CR.read() |0x10000000); /* 1 */
    else
        CR.write(CR.read() & 0xEFFFFFFF); /* 0 */
}

//Compute XER overflow fields SO, OV
//Arguments:
//int result -> The result register
//int s1 -> Source 1
//int s2 -> Source 2
//int s3 -> Source 3 (if only two sources, use 0)
inline void add_XER_OV_SO_update(int result,int s1,int s2,int s3) {

    long long int longresult =
        (long long int)(int)s1 + (long long int)(int)s2 + (long long int)(int)s3;

    if(longresult != (long long int)(int)result) {
        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    }
    else {
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    }
}

//Compute XER carry field CA
//Arguments:
//int result -> The result register
//int s1 -> Source 1
//int s2 -> Source 2
//int s3 -> Source 3 (if only two sources, use 0)
inline void add_XER_CA_update(int result,int s1,int s2,int s3) {

```

```

unsigned long long int longresult =
    (unsigned long long int)(unsigned int)s1 +
    (unsigned long long int)(unsigned int)s2 +
    (unsigned long long int)(unsigned int)s3;

if(longresult > 0xFFFFFFFF)
    XER.write(XER.read() | 0x20000000); /* Write 1 to bit 2 CA */
else
    XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */

}

//Compute XER overflow fields SO, OV
//Arguments:
//int result -> The result register
//int s1 -> Source 1
//int s2 -> Source 2
inline void divws_XER_OV_SO_update(int result,int s1,int s2) {

    long long int longresult =
        (long long int)(int)s1 / (long long int)(int)s2;

    if(longresult != (long long int)(int)result) {
        XER.write(XER.read() | 0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() | 0x80000000); /* Write 1 to bit 0 SO */
    }
    else
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */

}

//Compute XER overflow fields SO, OV
//Arguments:
//int result -> The result register

```

```
//int s1 -> Source 1
//int s2 -> Source 2
inline void divwu_XER_OV_SO_update(int result,int s1,int s2) {

    unsigned long long int longresult =
        (unsigned long long int)(unsigned int)s1 /
        (unsigned long long int)(unsigned int)s2;

    if(longresult != (unsigned long long int)(unsigned int)result) {
        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    }
    else
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */

}

//Ceil function
inline int ceil(int value, int divisor) {
    int res;
    if ((value % divisor)!=0)
        res=int(value/divisor)+1;
    else res=value/divisor;
    return res;
}

//Rotl function
inline unsigned int rotl(unsigned int reg,unsigned int n) {
    unsigned int tmp1=reg;
    unsigned int tmp2=reg;
    unsigned int rotated=(tmp1 << n) |(tmp2 >> 32-n);

    return(rotated);
}
```

```
//Mask32rlw function
inline unsigned int mask32rlw(unsigned int i,unsigned int f) {

    unsigned int maski,maskf;

    if(i<=f) {
        maski=(0xFFFFFFFF>>i);
        maskf=(0xFFFFFFFF<<(31-f));
        return(maski & maskf);
    }
    else {
        maski=(0xFFFFFFFF>>f);
        maski=maski>>1;
        maskf=(0xFFFFFFFF<<(31-i));
        maskf=maskf<<1;
        return(~(maski & maskf));
    }
}

//Function that returns the value of XER TBC
inline unsigned int XER_TBC_read() {
    return(XER.read() & 0x0000007F);
}

//Function that returns the value of XER OV
inline unsigned int XER_OV_read() {
    if(XER.read() & 0x40000000)
        return 1;
    else
        return 0;
}

//Function that returns the value of XER SO
```

```
inline unsigned int XER_SO_read() {
    if(XER.read() & 0x80000000)
        return 1;
    else
        return 0;
}

//Function that returns the value of XER CA
inline unsigned int XER_CA_read() {
    if(XER.read() & 0x20000000)
        return 1;
    else
        return 0;
}

//Function dump various registers
inline void dumpREG() {
    dbg_printf("XER.OV = %d\n",XER_OV_read());
    dbg_printf("XER.SO = %d\n",XER_SO_read());
    dbg_printf("XER.CA = %d\n",XER_CA_read());
    dbg_printf("CR = %#x\n",CR.read());
    dbg_printf("LR = %#x\n",LR.read());
    dbg_printf("CTR = %#x\n",CTR.read());
}

enum macType { NORMAL , NEGATIVE , MULTIPLY };

enum macOrder { CROSS , HIGH , LOW };

enum macOverflow { MODULE , SATURATE };

enum macSign { SIGNED , UNSIGNED };
```



```

ac_word genericMac ( macType   tp ,
                    macOrder   od ,
                    macOverflow of ,
                    macSign    sn ,
                    ac_Uword    oe ,
                    ac_Uword    rc ,
                    ac_word     data1 ,
                    ac_word     data2 ,
                    ac_word     data3 )
{
    ac_UHword value1 = 0; // first halfword operand
    ac_UHword value2 = 0; // second halfword operand
    ac_Uword  result = 0; // result word
    ac_Uword  prod   = 0;
    ac_UDword longResult = 0;

    switch ( od )
    {
        case CROSS:
            value1 = (ac_UHword)(data1 & 0x0000FFFF); // low-order halfword
            value2 = (ac_UHword)(data2 >> 16);       // high-order halfword
            break;
        case HIGH:
            value1 = (ac_UHword)(data1 >> 16);       // high-order halfword
            value2 = (ac_UHword)(data2 >> 16);       // high-order halfword
            break;
        case LOW:
            value1 = (ac_UHword)(data1 & 0x0000FFFF); // low-order halfword
            value2 = (ac_UHword)(data2 & 0x0000FFFF); // low-order halfword
            break;
    }

    switch ( sn )
    {
        case SIGNED:

```

```
        prod = (ac_Sword)((ac_SHword)value1) * (ac_Sword)((ac_SHword)value2);
        break;
    case UNSIGNED:
        prod = (ac_Uword)value1 * (ac_Uword)value2;
        break;
}

switch ( tp )
{
    case NORMAL:
        longResult = (ac_SDword)((ac_Sword)data3) + (ac_SDword)((ac_Sword)prod);
        break;
    case NEGATIVE:
        longResult = (ac_SDword)((ac_Sword)data3) - (ac_SDword)((ac_Sword)prod);
        break;
    case MULTIPLY:
        longResult = prod;
        break;
}

if ( of == SATURATE)
{
    ac_SDword smaxValue = 0x80000000; // 2^31

    ac_UDword umaxValue = 0xFFFFFFFF; // 2^32-1

    switch ( sn )
    {
        case SIGNED:
            if ((ac_SDword)longResult < -smaxValue )
                longResult = -smaxValue;
            if ((ac_SDword)longResult > smaxValue-1 )
                longResult = smaxValue-1;
            break;
        case UNSIGNED:
```

```
        if ( longResult > umaxValue )
            longResult = umaxValue;
        break;
    }
}

result = longResult;

if ( oe == 1 )
{
    if(longResult != (ac_UDword)result)
    {
        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    } else {
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    }
}

if ( rc == 1 )
{
    CR0_update(result);
}

return result;
}

/* Checking forwarding for a register */
inline ac_Uword forwarding(ac_Uword reg, ac_Uword value)
{
    ac_Uword result;

    if ( (EX_WB.regwrite == 1) && (EX_WB.rdest == reg) )
    {
        result = EX_WB.alures; // forwarding from EX_WB
    }
}
```

```

    dbg_printf("\n FORWARDING EX_WB: register %d with value %#x\n",reg,result);
}
else if ( (WB_LW.ldwrite == 1) && (WB_LW.lddest == reg) )
{
    result = WB_LW.lddata; // forwarding from WB_LW
    dbg_printf("\n FORWARDING WB_LW: register %d with value %#x\n",reg,result);
}
else
    result = value; // not forwarding

return result;
}

```

//!Generic instruction behavior method.

```

void ac_behavior( instruction )
{
    switch( stage ) {
    case FE: {
        dbg_printf("\n** FETCH **\n");

        dbg_printf("\n program counter=%#x\n", (ac_Uword)ac_pc);
        ac_pc += 4;
        FE.DE.npc = ac_pc;

        } break;

    case DE: {
        dbg_printf("\n** DECODE ** npc=%#x\n", (ac_Uword)FE.DE.npc);

        /* Instructions has to enable this registers when required */
        DE.EX.regwrite = 0;
        DE.EX.ldwrite = 0;

        } break;

```

```

case EX: {
    dbg_printf("\n** EXECUTE ** npc=%#x data1=%#x data2=%#x data3=%#x ra=%#x
rb=%#x r3=%#x mb=%#x me=%#x sh=%#x nb=%#x u3a=%#x u3b=%#x rf=%#x regwrite=%#x
oe=%#x rc=%#x ui=%#x si=%#x d=%#x ldwrite=%#x stwrite=%#x bo=%#x bi=%#x bd=%#x
aa=%#x lk=%#x li=%#x \n", (ac_Uword)DE_EX.npc, (ac_Uword)DE_EX.data1, (ac_Uword)DE_EX
(ac_Uword)DE_EX.data3, (ac_Uword)DE_EX.ra, (ac_Uword)DE_EX.rb, (ac_Uword)DE_EX.r3,
(ac_Uword)DE_EX.mb, (ac_Uword)DE_EX.me, (ac_Uword)DE_EX.sh, (ac_Uword)DE_EX.nb,
(ac_Uword)DE_EX.u3a, (ac_Uword)DE_EX.u3b, (ac_Uword)DE_EX.rf, (ac_Uword)DE_EX.regwrit
(ac_Uword)DE_EX.oe, (ac_Uword)DE_EX.rc, (ac_Uword)DE_EX.ui, (ac_word)DE_EX.si,
(ac_word)DE_EX.d, (ac_Uword)DE_EX.ldwrite, (ac_Uword)DE_EX.stwrite, (ac_Uword)DE_EX.b
(ac_Uword)DE_EX.bi, (ac_word)DE_EX.bd, (ac_Uword)DE_EX.aa, (ac_Uword)DE_EX.lk,
(ac_word)DE_EX.li);

    /* Instructions has to enable this registers when required */
    EX_WB.regwrite = 0;
    EX_WB.stwrite = 0;
    EX_WB.ldwrite = 0;

    } break;

case WB: {
    dbg_printf("\n** WRITE BACK ** alures=%#x wdata=%#x ea=%#x rdest=%#x regwrite=%#
stwrite=%#x ldwrite=%#x lddest=%#x \n", (ac_Uword)EX_WB.alures, (ac_Uword)EX_WB.wdat
(ac_Uword)EX_WB.ea, (ac_Uword)EX_WB.rdest, (ac_Uword)EX_WB.regwrite, (ac_Uword)EX_WB.
(ac_Uword)EX_WB.ldwrite, (ac_Uword)EX_WB.lddest);

    /* Execute write back when allowed */
    if (EX_WB.regwrite == 1)
    {
        dbg_printf("\n== writing GPR[%#x]=%#x \n", (ac_Uword)EX_WB.rdest, (ac_Uword)EX
        GPR.write(EX_WB.rdest, EX_WB.alures);
    }

    WB_LW.ldwrite = 0;

```

```
    } break;

    case LW: {
        dbg_printf("\n** LOAD WRITE BACK ** lddata=%#x lddest=%#x ldwrite=%#x \n",
            (ac_Uword)WB_LW.lddata, (ac_Uword)WB_LW.lddest, (ac_Uword)WB_LW.ldwrite);

        /* Execute load write back when allowed */
        if (WB_LW.ldwrite == 1)
        {
            dbg_printf("\n== writing GPR[%#x]=%#x \n", (ac_Uword)WB_LW.lddest, (ac_Uword)WB_LW.lddata);
            GPR.write(WB_LW.lddest, WB_LW.lddata);
        }

        } break;

    default: {
        } break;
    }
};

//!Generic begin behavior method.
void ac_behavior( begin )
{
    dbg_printf("Starting simulator...\n");

    /* Here the stack is started in a */
    GPR[1] = AC_RAM_END - 1024;

    /* Make a jump out of memory if it doesn't have an abi */
    LR.write(0xFFFFFFFF);
}
```

```
//! Instruction Format behavior methods.
void ac_behavior( I1 ){}

void ac_behavior( B1 ){}

void ac_behavior( SC1 ){}

void ac_behavior( D1 ){
    switch( stage ) {
        case FE: {
            } break;

        case DE: {

            DE_EX.ra = ra;
            DE_EX.r3 = rt;
            DE_EX.d = d;

            /* Checking forwarding for the ra register
            if ( (EX_WB.regwrite == 1) && (EX_WB.rdest == DE_EX.ra) )
                DE_EX.data1 = EX_WB.alures;
            else if ( (WB_LW.ldwrite == 1) && (WB_LW.lddest == DE_EX.ra) )
                DE_EX.data1 = WB_LW.lddata;
            else
                DE_EX.data1 = GPR.read(DE_EX.ra);*/

            DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

            } break;

        case EX: {

            /* Checking forwarding for the ra register
            if ( (EX_WB.regwrite == 1) && (EX_WB.rdest == DE_EX.ra) )
```

```
    ex_value1 = EX.WB.alures;
else if ( (WB.LW.ldwrite == 1) && (WB.LW.lddest == DE.EX.ra) )
    ex_value1 = WB.LW.lddata;
else
    ex_value1 = DE.EX.data1;*/

ex_value1 = forwarding( DE.EX.ra , DE.EX.data1 );

} break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( D2 ){}

void ac_behavior( D3 ){
    switch( stage ) {
        case FE: {
            } break;

        case DE: {

            DE.EX.ra = ra;
            DE.EX.r3 = rs;
            DE.EX.d = d;

            DE.EX.stwrite = 1;
```



```
    DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

    DE_EX.data2 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

    } break;

case EX: {

    ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

    ex_value2 = forwarding( DE_EX.r3 , DE_EX.data2 ); // forwarding rs

    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( D4 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {
```

```
DE_EX.ra = ra;
DE_EX.r3 = rs;
DE_EX.ui = ui;

DE_EX.regwrite = 1;

DE_EX.data1 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

} break;

case EX: {

    ex_value1 = forwarding( DE_EX.r3 , DE_EX.data1 ); // forwarding rs

} break;

case WB: {
} break;

case LW: {
} break;

default: {
} break;
}
}

void ac_behavior( D5 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {
```

```
DE_EX.u3a = bf;
DE_EX.ra = ra;
DE_EX.si = si;

DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

} break;

case EX: {

    ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

} break;

case WB: {
} break;

case LW: {
} break;

default: {
} break;
}
}

void ac_behavior( D6 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.u3a = bf;
        DE_EX.ra = ra;
```

```
    DE_EX.ui = ui;

    DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

    } break;

case EX: {

    ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( D7 ){}
void ac_behavior( X1 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
        DE_EX.rb = rb;
        DE_EX.r3 = rt;
```

```
DE_EX.regwrite = 1;
DE_EX.rc = rc;

DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb

} break;

case EX: {

ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb

} break;

case WB: {
} break;

case LW: {
} break;

default: {
} break;
}
}

void ac_behavior( X2 ){
switch( stage ) {
case FE: {
} break;
```

```
case DE: {

    DE_EX.ra = ra;
    DE_EX.rb = rb;
    DE_EX.r3 = rt;

    DE_EX.ldwrite = 1;

    DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

    DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb

} break;

case EX: {

    ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

    ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb

} break;

case WB: {
} break;

case LW: {
} break;

default: {
} break;
}
}
```

```
void ac_behavior( X3 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
        DE_EX.nb = nb;
        DE_EX.r3 = rt;

        DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

        } break;

    case EX: {

        ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

        } break;

    case WB: {
        } break;

    case LW: {
        } break;

    default: {
        } break;
    }
}

void ac_behavior( X4 ){}
```

```
void ac_behavior( X5 ){}
```

```
void ac_behavior( X6 ){  
    switch( stage ) {  
        case FE: {  
            } break;  
  
        case DE: {  
  
            DE_EX.r3 = rt;  
            DE_EX.regwrite = 1;  
  
            } break;  
  
        case EX: {  
            } break;  
  
        case WB: {  
            } break;  
  
        case LW: {  
            } break;  
  
        default: {  
            } break;  
        }  
    }  
}
```

```
void ac_behavior( X7 ){  
    switch( stage ) {  
        case FE: {  
            } break;  
  
        case DE: {
```



```
DE_EX.ra = ra;
DE_EX.rb = rb;
DE_EX.r3 = rs;

DE_EX.regwrite = 1;
DE_EX.rc = rc;

DE_EX.data1 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb

} break;

case EX: {

ex_value1 = forwarding( DE_EX.r3 , DE_EX.data1 ); // forwarding rs

ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb

} break;

case WB: {
} break;

case LW: {
} break;

default: {
} break;
}
}
```

```
void ac_behavior( X8 ){}

void ac_behavior( X9 ){
    switch( stage ) {
        case FE: {
            } break;

        case DE: {

            DE_EX.ra = ra;
            DE_EX.rb = rb;
            DE_EX.r3 = rs;

            DE_EX.stwrite = 1;

            DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra
            DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb
            DE_EX.data3 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

            } break;

        case EX: {

            ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra
            ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb
            ex_value3 = forwarding( DE_EX.r3 , DE_EX.data3 ); // forwarding rs

            } break;
    }
}
```

```
case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( X10 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
        DE_EX.nb = nb;
        DE_EX.r3 = rs;

        DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

        } break;

    case EX: {

        ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

        } break;

    case WB: {
        } break;
```

```
case LW: {
    } break;

default: {
    } break;
}

}

void ac_behavior( X11 ){}

void ac_behavior( X12 ){
    switch( stage ) {
case FE: {
    } break;

case DE: {

    DE_EX.ra = ra;
    DE_EX.rb = sh;
    DE_EX.r3 = rs;

    DE_EX.regwrite = 1;
    DE_EX.rc = rc;

    DE_EX.data1 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

    } break;

case EX: {

    ex_value1 = forwarding( DE_EX.r3 , DE_EX.data1 ); // forwarding rs

    } break;
```

```
    case WB: {
        } break;

    case LW: {
        } break;

    default: {
        } break;
    }
}

void ac_behavior( X13 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
        DE_EX.r3 = rs;

        DE_EX.regwrite = 1;
        DE_EX.rc = rc;

        DE_EX.data1 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

        } break;

    case EX: {

        ex_value1 = forwarding( DE_EX.r3 , DE_EX.data1 ); // forwarding rs

        } break;
```

```
case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}

void ac_behavior( X14 ){}

void ac_behavior( X15 ){}

void ac_behavior( X16 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
        DE_EX.rb = rb;
        DE_EX.u3a = bf;

        DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

        DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb

    } break;
```

```
case EX: {

    ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

    ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb

    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}

}

void ac_behavior( X17 ){}

void ac_behavior( X18 ){}

void ac_behavior( X19 ){}

void ac_behavior( X20 ){}

void ac_behavior( X21 ){}

void ac_behavior( X22 ){}

void ac_behavior( X23 ){}

void ac_behavior( X24 ){}

```

```
void ac_behavior( X25 ){}
```

```
void ac_behavior( XL1 ){  
    switch( stage ) {  
        case FE: {  
            } break;  
  
        case DE: {  
  
            DE_EX.ra = ba;  
            DE_EX.rb = bb;  
            DE_EX.r3 = bt;  
  
            } break;  
  
        case EX: {  
            } break;  
  
        case WB: {  
            } break;  
  
        case LW: {  
            } break;  
  
        default: {  
            } break;  
        }  
    }  
}
```

```
void ac_behavior( XL2 ){}
```

```
void ac_behavior( XL3 ){  
    switch( stage ) {  
        case FE: {  
            } break;  
    }  
}
```



```
case DE: {

    DE_EX.u3a = bf;
    DE_EX.u3b = bfa;

    } break;

case EX: {
    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}

}

void ac_behavior( XL4 ){}

void ac_behavior( XFX1 ){
    switch( stage ) {
        case FE: {
            } break;

        case DE: {

            DE_EX.r3 = rt;
            DE_EX.rf = sprf;

            DE_EX.regwrite = 1;
        }
    }
}
```

```
    } break;

case EX: {
    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}

}

void ac_behavior( XFX2 ){}

void ac_behavior( XFX3 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.r3 = rs;
        DE_EX.rf = xfm;

        DE_EX.data1 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

        } break;

    case EX: {
```

```
ex_value1 = forwarding( DE_EX.r3 , DE_EX.data1 ); // forwarding rs

    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( XFX4 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.r3 = rs;
        DE_EX.rf = sprf;

        DE_EX.data1 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

        } break;

    case EX: {

        ex_value1 = forwarding( DE_EX.r3 , DE_EX.data1 ); // forwarding rs
```

```
    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( XFX5 ){}

void ac_behavior( X01 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
        DE_EX.rb = rb;
        DE_EX.r3 = rt;

        DE_EX.regwrite = 1;
        DE_EX.oe = oe;
        DE_EX.rc = rc;

        DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

        DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb
```

```
    DE_EX.data3 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rt

    } break;

case EX: {

    ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

    ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb

    ex_value3 = forwarding( DE_EX.r3 , DE_EX.data3 ); // forwarding rt

    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( X02 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
```

```
DE_EX.rb = rb;
DE_EX.r3 = rt;

DE_EX.regwrite = 1;
DE_EX.rc = rc;

DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb

} break;

case EX: {

ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb

} break;

case WB: {
} break;

case LW: {
} break;

default: {
} break;
}
}

void ac_behavior( X03 ){
switch( stage ) {
```

```
case FE: {
    } break;

case DE: {

    DE_EX.ra = ra;
    DE_EX.r3 = rt;

    DE_EX.regwrite = 1;
    DE_EX.oe = oe;
    DE_EX.rc = rc;

    DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra

    } break;

case EX: {

    ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra

    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

void ac_behavior( M1 ){
```

```
switch( stage ) {
case FE: {
    } break;

case DE: {

    DE_EX.ra = ra;
    DE_EX.rb = rb;
    DE_EX.r3 = rs;
    DE_EX.mb = mb;
    DE_EX.me = me;

    DE_EX.regwrite = 1;
    DE_EX.rc = rc;

    DE_EX.data1 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

    DE_EX.data2 = forwarding( DE_EX.rb , GPR.read(DE_EX.rb) ); // forwarding
rb

    } break;

case EX: {

    ex_value1 = forwarding( DE_EX.r3 , DE_EX.data1 ); // forwarding rs

    ex_value2 = forwarding( DE_EX.rb , DE_EX.data2 ); // forwarding rb

    } break;

case WB: {
    } break;

case LW: {
```



```
    } break;

default: {
    } break;
}
}

void ac_behavior( M2 ){
    switch( stage ) {
    case FE: {
        } break;

    case DE: {

        DE_EX.ra = ra;
        DE_EX.r3 = rs;
        DE_EX.sh = sh;
        DE_EX.mb = mb;
        DE_EX.me = me;

        DE_EX.regwrite = 1;
        DE_EX.rc = rc;

        DE_EX.data1 = forwarding( DE_EX.ra , GPR.read(DE_EX.ra) ); // forwarding
ra
        DE_EX.data2 = forwarding( DE_EX.r3 , GPR.read(DE_EX.r3) ); // forwarding
rs

        } break;

    case EX: {

        ex_value1 = forwarding( DE_EX.ra , DE_EX.data1 ); // forwarding ra
```

```
ex_value2 = forwarding( DE_EX.r3 , DE_EX.data2 ); // forwarding rs

    } break;

case WB: {
    } break;

case LW: {
    } break;

default: {
    } break;
}
}

//!Instruction add behavior method.
// Add
void ac_behavior( add )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" add%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = ex_value1 + ex_value2;
```

```

EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.r3;

if (DE_EX.oe==1)
    add_XER_OV_SO_update(EX_WB.alures,ex_value1,ex_value2,0);
if (DE_EX.rc==1)
    CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction addc behavior method.
// Add Carrying
void ac_behavior( addc )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" addc%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r

        } break;

```

```
case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = ex_value1 + ex_value2;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    add_XER_CA_update(EX_WB.alures,ex_value1, ex_value2,0);

    if (DE_EX.oe==1)
        add_XER_OV_SO_update(EX_WB.alures,ex_value1,ex_value2,0);
    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction adde behavior method.
// Add Extended
```

```
void ac_behavior( adde )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" adde%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        EX_WB.alures = ex_value1 + ex_value2 + XER_CA_read();
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        add_XER_CA_update(EX_WB.alures, ex_value1, ex_value2, XER_CA_read());

        if (DE_EX.oe==1)
            add_XER_OV_SO_update(EX_WB.alures, ex_value1, ex_value2, XER_CA_read())
        if (DE_EX.rc==1)
            CR0_update(EX_WB.alures);

        } break;

    case WB: {
        dbg_printf(" %s \n", get_name());
        } break;

    case LW: {
        dbg_printf(" %s \n", get_name());
```

```
    } break;

default: {
    } break;
}
};

//!Instruction addi behavior method.
// Add Immediate
void ac_behavior( addi )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" addi r%d, r%d, %d\n\n",rt,ra,d);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ime32 = DE_EX.d;

        if (DE_EX.ra == 0)
            EX.WB.alures = ime32;
        else
            EX.WB.alures = ex.value1 + ime32;
```

```
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.r3;

} break;

case WB: {
    dbg_printf(" %s \n", get_name());
} break;

case LW: {
    dbg_printf(" %s \n", get_name());
} break;

default: {
} break;
};

//!Instruction addic behavior method.
// Add Immediate Carrying
void ac_behavior( addic )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" addic r%d, r%d, %d\n\n",rt,ra,d);

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());

        DE_EX.regwrite = 1;

        } break;
```

```
case EX: {
    dbg_printf(" %s \n",get_name());

    int ime32 = DE_EX.d;

    int result = ex_value1 + ime32;

    add_XER_CA_update(result,ex_value1,ime32,0);

    EX_WB.alures = result;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction addic_behavior method.
// Add Immediate Carrying and Record
void ac_behavior( addic_)
{
    switch( stage ) {
    case FE: {
```



```
    dbg_printf(" addic.  r%d, r%d, %d\n\n",rt,ra,d);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());

    DE_EX.regwrite = 1;

    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ime32 = DE_EX.d;

    int result = ex_value1 + ime32;

    add_XER_CA_update(result,ex_value1,ime32,0);

    CR0_update(result); // Do not have rc field and update CR0

    EX_WB.alures = result;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
```

```
    } break;

default: {
    } break;
}
};

//!Instruction addis behavior method.
// Add Immediate Shifted
void ac_behavior( addis )
{
    switch( stage ) {
    case FE: {
        dbg_printf(" addis r%d, r%d, %d\n\n",rt,ra,d);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ime32 = DE_EX.d;

        ime32 = ime32 << 16;

        if (DE_EX.ra == 0)
            EX_WB.alures = ime32;
        else
```

```
    EX_WB.alures = ex.value1 + ime32;

    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction addme behavior method.
// Add to Minus One Extended
void ac_behavior( addme )
{
    switch( stage ) {
    case FE: {
        dbg_printf(" addme%s r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt,ra);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
```

```

    dbg_printf(" %s \n",get_name());

    EX_WB.alures  = ex_value1 + XER_CA.read() + (-1);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest   = DE_EX.r3;

    add_XER_CA_update(EX_WB.alures, ex_value1, XER_CA.read(), -1);

    if (DE_EX.oe==1)
        add_XER_OV_SO_update(EX_WB.alures, ex_value1, XER_CA.read(), -1);
    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction addze behavior method.
// Add to Zero Extended
void ac_behavior( addze )
{
    switch( stage ) {
    case FE: {

```

```
    dbg_printf(" addze%s r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), rt, ra);

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = ex_value1 + XER_CA_read();
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    add_XER_CA_update(EX_WB.alures, ex_value1, XER_CA_read(), 0);

    if (DE_EX.oe==1)
        add_XER_OV_SO_update(EX_WB.alures, ex_value1, XER_CA_read(), 0);
    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
```

```
};

//!Instruction ande behavior method.
// AND
void ac_behavior( ande )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" and%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);

            } break;

        case DE: {
            dbg_printf(" %s \n", get_name());
            } break;

        case EX: {
            dbg_printf(" %s \n", get_name());

            EX_WB.alures = ex_value1 & ex_value2;
            EX_WB.regwrite = DE_EX.regwrite;
            EX_WB.rdest = DE_EX.ra;

            if (DE_EX.rc==1)
                CR0_update(EX_WB.alures);

            } break;

        case WB: {
            dbg_printf(" %s \n", get_name());
            } break;

        case LW: {
            dbg_printf(" %s \n", get_name());
```

```
    } break;

default: {
    } break;
}
};

//!Instruction andc behavior method.
// AND with Complement
void ac_behavior( andc )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" andc%s r%d, r%d, r%d\n\n", (rc==1?".":""), ra,rs,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = ex.value1 & ~ex.value2;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;

        if (DE_EX.rc==1)
            CRO_update(EX_WB.alures);

        } break;

    case WB: {
```

```
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction andi_behavior method.
// AND Immediate
void ac_behavior( andi_)
{
    switch( stage ) {
    case FE: {

        dbg_printf(" andi.  r%d, r%d, %d\n\n",ra,rs,ui);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int ime32 = (unsigned short int)DE_EX.ui;

        int result = ex_value1 & ime32;

        CR0_update(result); // Do not have rc field and update CR0
```



```
EX_WB.alures = result;
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.ra;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction andis_behavior method.
// AND Immediate Shifted
void ac_behavior( andis_)
{
    switch( stage ) {
    case FE: {

        dbg_printf(" andis.  r%d, r%d, %d\n\n",ra,rs,ui);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;
    }
```

```
case EX: {
    dbg_printf(" %s \n",get_name());

    unsigned int ime32 = (unsigned short int)DE_EX.ui;

    ime32 = ime32 << 16;

    int result = ex_value1 & ime32;

    CR0_update(result); // Do not have rc field and update CR0

    EX_WB.alures = result;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction b behavior method.
// Branch
void ac_behavior( b )
{
    switch( stage ) {
```

```
case FE: {

    dbg_printf(" b%s %d\n\n", (lk==1?(aa==1?"la":"l"):(aa==1?"a":"")), li);
    ac_pc = 0x100;

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());

    DE_EX.npc = FE.DE.npc;
    DE_EX.lk = lk;
    DE_EX.aa = aa;
    DE_EX.li = li;

    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    int displacement;
    unsigned int nia;

    displacement = DE_EX.li << 2;

    nia = displacement;

    if (DE_EX.aa==0)
        nia = nia + (DE_EX.npc-4); /* Because pre-increment */

    ac_pc = delay(nia,0);
    dbg_printf("\n BRANCH TAKEN \n");

    ac_flush("FE");          dbg_printf("\n FLUSH FE \n");
    ac_flush("DE");          dbg_printf("\n FLUSH DE \n");
```

```
    if (DE_EX.lk==1)
        LR.write(DE_EX.npc);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction bc behavior method.
// Branch Conditional
void ac_behavior( bc )
{
    switch( stage ) {
case FE: {
    dbg_printf(" bc%s %d, %d, %d\n\n", (lk==1?(aa==1?"la":"l"):(aa==1?"a":"")),bo,bi,l

    ac_pc = 0x100;

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());

    DE_EX.npc = FE_DE.npc;
```

```
DE_EX.bo = bo;
DE_EX.bi = bi;
DE_EX.bd = bd;
DE_EX.aa = aa;
DE_EX.lk = lk;

} break;

case EX: {
    dbg_printf(" %s \n", get_name());

    int displacement;
    unsigned int nia;

    unsigned int masc = 0x80000000 >> DE_EX.bi;

    if ((DE_EX.bo & 0x04) == 0x00) /* decrement CTR */
        CTR.write(CTR.read()-1);

    dbg_printf("\n      CR=%#x \n", CTR.read());
    dbg_printf("\n      CTR=%#x \n", CTR.read());

    if (((DE_EX.bo & 0x04) || /* Branch */
        ((CTR.read()==0) && (DE_EX.bo & 0x02)) ||
        (!(CTR.read()==0) && !(DE_EX.bo & 0x02)))
        &&
        ((bo & 0x10) ||
         (((CR.read() & masc) && (DE_EX.bo & 0x08)) ||
          (!(CR.read() & masc) && !(DE_EX.bo & 0x08))))) {

        dbg_printf("\n BRANCH TAKEN \n");

        displacement = DE_EX.bd << 2;
```

```
    nia = displacement;

    if (DE_EX.aa == 0)
        nia = nia + (DE_EX.npc-4); /* because pre-increment */

    ac_pc = delay(nia,0); dbg_printf("\n PC=%#x \n",nia);
    ac_flush("FE");      dbg_printf("\n FLUSH FE \n");
    ac_flush("DE");      dbg_printf("\n FLUSH DE \n");
}
else {
    dbg_printf("\n BRANCH NOT TAKEN \n");
    ac_pc = delay(DE_EX.npc,0); dbg_printf("\n PC=%#x \n", (ac_Uword) DE_EX.npc);
    ac_flush("FE"); dbg_printf("\n FLUSH FE \n");
    ac_flush("DE"); dbg_printf("\n FLUSH DE \n");

}

if (DE_EX.lk==1) /* save PC to LR */
    LR.write(DE_EX.npc);

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};
```

```
//!Instruction bcctr behavior method.
// Branch Conditional to Count Register
void ac_behavior( bcctr )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" bcctr%s %d, %d\n\n", (lk==1?"1":""), bo, bi);

        ac_pc = 0x100;

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());

        DE_EX.npc = FE.DE.npc;

        DE_EX.bo = bo;
        DE_EX.bi = bi;
        DE_EX.lk = lk;

        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        unsigned int masc = 0x80000000 >> DE_EX.bi;

        if ((DE_EX.bo & 0x04) == 0x00) /* decrement CTR */
            CTR.write(CTR.read()-1);

        if (((DE_EX.bo & 0x04) || /* Branch */
            ((CTR.read()==0) && (DE_EX.bo & 0x02))) ||
```

```

        (!(CTR.read()==0) && !(DE_EX.bo & 0x02)))
        &&
        ((DE_EX.bo & 0x10) ||
        (((CR.read() & masc) && (DE_EX.bo & 0x08)) ||
        (!(CR.read() & masc) && !(DE_EX.bo & 0x08)))))) {

    dbg_printf("\n BRANCH TAKEN \n");

    ac_pc = delay(CTR.read() & 0xFFFFFFFF, 0); dbg_printf("\n PC=%#x \n",CTR.read
    & 0xFFFFFFFF);
    ac_flush("FE"); dbg_printf("\n FLUSH FE \n");
    ac_flush("DE"); dbg_printf("\n FLUSH DE \n");
}
else {
    dbg_printf("\n BRANCH NOT TAKEN \n");
    ac_pc = delay(DE_EX.npc,0); dbg_printf("\n PC=%#x \n", (ac_Uword)
DE_EX.npc);
    ac_flush("FE"); dbg_printf("\n FLUSH FE \n");
    ac_flush("DE"); dbg_printf("\n FLUSH DE \n");
}

if (DE_EX.lk==1) /* save PC to LR */
    LR.write(DE_EX.npc);

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

```



```
default: {
    } break;
}
};

//!Instruction bclr behavior method.
// Branch Conditional to Link Register
void ac_behavior( bclr )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" bclr%s %d, %d\n\n", (lk==1?"1":""), bo, bi);

        ac_pc = 0x100;

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());

        DE_EX.npc = FE.DE.npc;

        DE_EX.bo = bo;
        DE_EX.bi = bi;
        DE_EX.lk = lk;

        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        unsigned int masc = 0x80000000 >> DE_EX.bi;

        if ((DE_EX.bo & 0x04) == 0x00) /* decrement CTR */
```

```

    CTR.write(CTR.read()-1);

    if (((DE_EX.bo & 0x04) || /* Branch */
        ((CTR.read()==0) && (DE_EX.bo & 0x02)) ||
        (!(CTR.read()==0) && !(DE_EX.bo & 0x02)))
        &&
        ((DE_EX.bo & 0x10) ||
        (((CR.read() & masc) && (DE_EX.bo & 0x08)) ||
        (!(CR.read() & masc) && !(DE_EX.bo & 0x08))))) {

        dbg_printf("\n BRANCH TAKEN \n");

        ac_pc = delay(LR.read() & 0xFFFFFFFFC , 0); dbg_printf("\n PC=%#x \n",LR.read()
& 0xFFFFFFFFC);
        ac_flush("FE"); dbg_printf("\n FLUSH FE \n");
        ac_flush("DE"); dbg_printf("\n FLUSH DE \n");
    }
    else {
        dbg_printf("\n BRANCH NOT TAKEN \n");
        ac_pc = delay(DE_EX.npc,0); dbg_printf("\n PC=%#x \n", (ac_Uword) DE_EX.npc);
        ac_flush("FE"); dbg_printf("\n FLUSH FE \n");
        ac_flush("DE"); dbg_printf("\n FLUSH DE \n");
    }

    if (DE_EX.lk==1) /* save PC to LR */
        LR.write(DE_EX.npc);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {

```

```
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction cmp behavior method.
// Compare
void ac_behavior( cmp )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" cmp crf%d, 0, r%d, r%d\n\n",bf,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int c = 0x00;
        unsigned int n = DE_EX.u3a; // bf
        unsigned int masc = 0xF0000000;

        if((int)ex_value1 < (int)ex_value2)
            c = c |0x80000000;
        if((int)ex_value1 > (int)ex_value2)
            c = c |0x40000000;
        if((int)ex_value1 == (int)ex_value2)
```

```
    c = c |0x20000000;
if(XER_SO_read()==1)
    c = c |0x10000000;

c = c >> (n*4);
masc = ~(masc >> (n*4));

CR.write((CR.read() & masc) |c);

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

//!Instruction cmpi behavior method.
// Compare Immediate
void ac_behavior( cmpi )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" cmpi crf%d, 0, r%d, %d\n\n",bf,ra,si);

    } break;
```

```
case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    unsigned int c = 0x00;
    unsigned int n = DE_EX.u3a; // bf
    unsigned int masc = 0xF0000000;

    int ime32 = (short int)(DE_EX.si);

    if((int)ex_value1 < ime32)
        c = c |0x80000000;
    if((int)ex_value1 > ime32)
        c = c |0x40000000;
    if((int)ex_value1 == ime32)
        c = c |0x20000000;
    if(XER_SO_read()==1)
        c = c |0x10000000;

    c = c >> (n*4);
    masc = ~(masc >> (n*4));

    CR.write((CR.read() & masc) |c);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
```

```
    } break;

default: {
    } break;
}
};

//!Instruction cmpl behavior method.
// Compare Logical
void ac_behavior( cmpl )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" cmpl crf%d, 0, r%d, r%d\n\n",bf,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int c = 0x00;
        unsigned int n = DE_EX.u3a; // bf
        unsigned int masc = 0xF0000000;

        unsigned int uintra = ex_value1;
        unsigned int uintrb = ex_value2;

        if(uintra < uintrb)
            c = c |0x80000000;
        if(uintra > uintrb)
```

```

        c = c |0x40000000;
    if(uintra == uintrb)
        c = c |0x20000000;
    if(XER_SO_read()==1)
        c = c |0x10000000;

    c = c >> (n*4);
    masc = ~(masc >> (n*4));

    CR.write((CR.read() & masc) |c);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction cmpli behavior method.
// Compare Logical Immediate
void ac_behavior( cmpli )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" cmpli crf%d, 0, r%d, %d\n\n",bf,ra,ui);
    }
    }
}

```

```
    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    unsigned int c = 0x00;
    unsigned int n = DE_EX.u3a; // bf
    unsigned int masc = 0xF0000000;

    unsigned int ime32 = (unsigned short int)(DE_EX.ui);

    if(ex_value1 < ime32)
        c = c | 0x80000000;
    if(ex_value1 > ime32)
        c = c | 0x40000000;
    if(ex_value1 == ime32)
        c = c | 0x20000000;
    if(XER_SO_read()==1)
        c = c | 0x10000000;

    c = c >> (n*4);
    masc = ~(masc >> (n*4));

    CR.write((CR.read() & masc) | c);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;
```



```
case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction cntlzw behavior method.
// Count Leading Zeros Word
void ac_behavior( cntlzw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" cntlzw%s r%d, r%d\n\n", (rc==1?" ":""),ra,rs);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int urs = ex_value1;
        unsigned int masc = 0x80000000;
        unsigned int n = 0;

        while(n < 32) {
            if(urs & masc)
                break;
            n++;
        }
    }
}
```

```
        masc = masc >> 1;
    }

    EX_WB.alures = n;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    if (DE_EX.rc==1)
        CR0_update(n);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction crand behavior method.
// Condition Register AND
void ac_behavior( crand )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" crand %d, %d, %d\n\n",bt,ba,bb);

        } break;
```

```
case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba = CR.read();
    CRbb = CR.read();

    /* Shift source bit to first position and zero another */
    CRba = (CRba << DE_EX.ra) & 0x80000000;
    CRbb = (CRbb << DE_EX.rb) & 0x80000000;

    CRbt = (CRba & CRbb) & 0x80000000;
    CRbt = CRbt >> DE_EX.r3;

    if(CRbt)
        CR.write(CR.read() | CRbt);
    else
        CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
```

```
    } break;

default: {
    } break;
}
};

//!Instruction crandc behavior method.
// Condition Register AND with Complement
void ac_behavior( crandc )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" crandc %d, %d, %d\n\n",bt,ba,bb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int CRbt;
        unsigned int CRba;
        unsigned int CRbb;

        CRba = CR.read();
        CRbb = CR.read();

        /* Shift source bit to first position and zero another */
        CRba = (CRba << DE_EX.ra) & 0x80000000;
        CRbb = (CRbb << DE_EX.rb) & 0x80000000;
```

```
CRbt = (CRba & ~CRbb) & 0x80000000;
CRbt = CRbt >> DE_EX.r3;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction creqv behavior method.
// Condition Register Equivalent
void ac_behavior( creqv )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" creqv %d, %d, %d\n\n",bt,ba,bb);

    } break;
```

```
case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba = CR.read();
    CRbb = CR.read();

    /* Shift source bit to first position and zero another */
    CRba = (CRba << DE_EX.ra) & 0x80000000;
    CRbb = (CRbb << DE_EX.rb) & 0x80000000;

    CRbt = ~(CRba ^CRbb) & 0x80000000;
    CRbt = CRbt >> DE_EX.r3;

    if(CRbt)
        CR.write(CR.read() |CRbt);
    else
        CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;
```

```
    default: {
        } break;
    }
};

//!Instruction crnand behavior method.
// Condition Register NAND
void ac_behavior( crnand )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" crnand %d, %d, %d\n\n",bt,ba,bb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int CRbt;
        unsigned int CRba;
        unsigned int CRbb;

        CRba = CR.read();
        CRbb = CR.read();

        /* Shift source bit to first position and zero another */
        CRba = (CRba << DE_EX.ra) & 0x80000000;
        CRbb = (CRbb << DE_EX.rb) & 0x80000000;
```

```
CRbt = ~(CRba & CRbb) & 0x80000000;
CRbt = CRbt >> DE_EX.r3;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction crnor behavior method.
// Condition Register NOR
void ac_behavior( crnor )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" crnor %d, %d, %d\n\n",bt,ba,bb);

        } break;

    case DE: {
```



```
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba = CR.read();
    CRbb = CR.read();

    /* Shift source bit to first position and zero another */
    CRba = (CRba << DE_EX.ra) & 0x80000000;
    CRbb = (CRbb << DE_EX.rb) & 0x80000000;

    CRbt = ~(CRba | CRbb) & 0x80000000;
    CRbt = CRbt >> DE_EX.r3;

    if(CRbt)
        CR.write(CR.read() | CRbt);
    else
        CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;
```

```
    default: {
        } break;
    }
};

//!Instruction cror behavior method.
// Condition Register OR
void ac_behavior( cror )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" cror %d, %d, %d\n\n",bt,ba,bb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int CRbt;
        unsigned int CRba;
        unsigned int CRbb;

        CRba = CR.read();
        CRbb = CR.read();

        /* Shift source bit to first position and zero another */
        CRba = (CRba << DE_EX.ra) & 0x80000000;
        CRbb = (CRbb << DE_EX.rb) & 0x80000000;

        CRbt = (CRba |CRbb) & 0x80000000;
```

```
CRbt = CRbt >> DE_EX.r3;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction crorc behavior method.
// Condition Register OR with Complement
void ac_behavior( crorc )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" crorc %d, %d, %d\n\n",bt,ba,bb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
```

```
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    unsigned int CRbt;
    unsigned int CRba;
    unsigned int CRbb;

    CRba = CR.read();
    CRbb = CR.read();

    /* Shift source bit to first position and zero another */
    CRba = (CRba << DE_EX.ra) & 0x80000000;
    CRbb = (CRbb << DE_EX.rb) & 0x80000000;

    CRbt = (CRba | ~CRbb) & 0x80000000;
    CRbt = CRbt >> DE_EX.r3;

    if(CRbt)
        CR.write(CR.read() | CRbt);
    else
        CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
```

```
    } break;
}

};

//!Instruction crxor behavior method.
// Condition Register XOR
void ac_behavior( crxor )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" crxor %d, %d, %d\n\n",bt,ba,bb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int CRbt;
        unsigned int CRba;
        unsigned int CRbb;

        CRba = CR.read();
        CRbb = CR.read();

        /* Shift source bit to first position and zero another */
        CRba = (CRba << DE_EX.ra) & 0x80000000;
        CRbb = (CRbb << DE_EX.rb) & 0x80000000;

        CRbt = (CRba ^ CRbb) & 0x80000000;
```

```
CRbt = CRbt >> DE_EX.r3;

if(CRbt)
    CR.write(CR.read() |CRbt);
else
    CR.write(CR.read() & ~(0x80000000 >> DE_EX.r3));

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction divw behavior method.
// Divide Word
void ac_behavior( divw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" divw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
```

```
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = (int)ex_value1 / (int)ex_value2;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    if (DE_EX.oe==1)
        divws_XER_OV_SO_update(EX_WB.alures, ex_value1, ex_value2);

    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

//!Instruction divwu behavior method.
// Divide Word Unsigned
void ac_behavior( divwu )
{
    switch( stage ) {
```

```
case FE: {

    dbg_printf(" divwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = (unsigned int)ex_value1 / (unsigned int)ex_value2;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    if (DE_EX.oe==1)
        divws_XER_OV_SO_update(EX_WB.alures, ex_value1, ex_value2);

    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
```



```
    }  
};  
  
//!Instruction eqv behavior method.  
// Equivalent  
void ac_behavior( eqv )  
{  
    switch( stage ) {  
    case FE: {  
  
        dbg_printf(" eqv%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);  
  
        } break;  
  
    case DE: {  
        dbg_printf(" %s \n", get_name());  
        } break;  
  
    case EX: {  
        dbg_printf(" %s \n", get_name());  
  
        EX_WB.alures = ~(ex_value1 ^ex_value2);  
        EX_WB.regwrite = DE_EX.regwrite;  
        EX_WB.rdest = DE_EX.ra;  
  
        if (DE_EX.rc==1)  
            CR0_update(EX_WB.alures);  
  
        } break;  
  
    case WB: {  
        dbg_printf(" %s \n", get_name());  
        } break;  
  
    case LW: {
```

```
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction extsb behavior method.
// Extend Sign Byte
void ac_behavior( extsb )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" extsb%s r%d, r%d\n\n", (rc==1?" ":""),ra,rs);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = (char)(ex_value1);
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;

        if (DE_EX.rc==1)
            CR0_update(EX_WB.alures);

        } break;
    }
```

```
case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction extsh behavior method.
// Extend Sign Halfword
void ac_behavior( extsh )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" extsh%s r%d, r%d\n\n", (rc==1?" ":""),ra,rs);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = (short int)(ex_value1);
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;
```

```
    if (DE.EX.rc==1)
        CRO_update(EX.WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lbz behavior method.
// Load Byte and Zero
void ac_behavior( lbz )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lbz r%d, %d(r%d)\n\n",rt,d,ra);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE.EX.ldwrite = 1;

        } break;
```

```
case EX: {
    dbg_printf(" %s \n", get_name());

    int ea;

    if (DE_EX.ra == 0)
        ea = (short int)DE_EX.d;
    else
        ea = ex_value1 + (short int)DE_EX.d;

    EX_WB.ea = ea;
    EX_WB.ldwrite = DE_EX.ldwrite;
    EX_WB.lddest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (unsigned int)MEM.read_byte(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};
```

```
//!Instruction lbzu behavior method.
// Load Byte and Zero with Update
void ac_behavior( lbzu )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" lbzu r%d, %d(r%d)\n\n",rt,d,ra);

            } break;

        case DE:{

            DE_EX.regwrite = 1;
            DE_EX.ldwrite = 1;

            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            int ea = ex_value1 + (short int)DE_EX.d;

            EX_WB.alures = ea;
            EX_WB.regwrite = DE_EX.regwrite;
            EX_WB.rdest = DE_EX.ra;

            EX_WB.ea = ea;
            EX_WB.ldwrite = DE_EX.ldwrite;
            EX_WB.lddest = DE_EX.r3;

            } break;

        case WB: {
            dbg_printf(" %s \n",get_name());
```

```
WB_LW.lddest = EX_WB.lddest;
WB_LW.lddata = (unsigned int)MEM.read_byte(EX_WB.ea);
WB_LW.ldwrite = EX_WB.ldwrite;

} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

//!Instruction lbzux behavior method.
// Load Byte and Zero with Update Indexed
void ac_behavior( lbzux )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lbzux r%d, r%d, r%d\n\n",rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;

        } break;

    case EX: {
```

```
    dbg_printf(" %s \n",get_name());

    int ea = ex_value1 + ex_value2;

    EX_WB.alures = ea;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    EX_WB.ea = ea;
    EX_WB.ldwrite = DE_EX.ldwrite;
    EX_WB.lddest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (unsigned int)MEM.read_byte(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lbzx behavior method.
// Load Byte and Zero Indexed
void ac_behavior( lbzx )
```



```
{
switch( stage ) {
case FE: {

    dbg_printf(" lbzx r%d, r%d, r%d\n\n",rt,ra,rb);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if (DE.EX.ra!=0)
        ea = ex_value1 + ex_value2;
    else
        ea = ex_value2;

    EX.WB.ea = ea;
    EX.WB.ldwrite = DE.EX.ldwrite;
    EX.WB.lddest = DE.EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB.LW.lddest = EX.WB.lddest;
    WB.LW.lddata = (unsigned int)MEM.read_byte(EX.WB.ea);
    WB.LW.ldwrite = EX.WB.ldwrite;
```

```
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lha behavior method.
// Load Halfword Algebraic
void ac_behavior( lha )
{
    switch( stage ) {
case FE: {

    dbg_printf(" lha r%d, %d(r%d)\n\n", rt, d, ra);

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());

    DE_EX.ldwrite = 1;

    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    int ea;

    if (DE_EX.ra != 0)
```

```
    ea = ex.value1 + (short int)DE_EX.d;
else
    ea = (short int)DE_EX.d;

EX_WB.ea = ea;
EX_WB.ldwrite = DE_EX.ldwrite;
EX_WB.lddest = DE_EX.r3;

} break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (short int)MEM.read_half(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

//!Instruction lhau behavior method.
// Load Halfword Algebraic with Update
void ac_behavior( lhau )
{
    switch( stage ) {
    case FE: {
```

```
dbg_printf(" lhau r%d, %d(r%d)\n\n",rt,d,ra);

} break;

case DE:{

DE_EX.regwrite = 1;
DE_EX.ldwrite = 1;

} break;

case EX: {
dbg_printf(" %s \n",get_name());

int ea = ex_value1 + (short int)DE_EX.d;

EX_WB.alures = ea;
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.ra;

EX_WB.ea = ea;
EX_WB.ldwrite = DE_EX.ldwrite;
EX_WB.lddest = DE_EX.r3;

} break;

case WB: {
dbg_printf(" %s \n",get_name());

WB_LW.lddest = EX_WB.lddest;
WB_LW.lddata = (short int)MEM.read_half(EX_WB.ea);
WB_LW.ldwrite = EX_WB.ldwrite;

} break;
```

```
case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}

};

//!Instruction lhaux behavior method.
// Load Halfword Algebraic with Update Indexed
void ac_behavior( lhaux )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lhaux r%d, r%d, r%d\n\n",rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea = ex_value1 + ex_value2;

        EX_WB.alures = ea;
        EX_WB.regwrite = DE_EX.regwrite;
    }
    }
}
```

```

EX_WB.rdest = DE_EX.ra;

EX_WB.ea = ea;
EX_WB.ldwrite = DE_EX.ldwrite;
EX_WB.lddest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (short int)MEM.read_half(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}

};

//!Instruction lhax behavior method.
// Load Halfword Algebraic Indexed
void ac_behavior( lhax )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lhax r%d, r%d, r%d\n\n",rt,ra,rb);

```

```
    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if (DE_EX.ra!=0)
        ea = ex_value1 + ex_value2;
    else
        ea = ex_value2;

    EX_WB.ea = ea;
    EX_WB.ldwrite = DE_EX.ldwrite;
    EX_WB.lddest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (short int)MEM.read_half(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;
```

```
    default: {
        } break;
    }
};

//!Instruction lhbrx behavior method.
// Load Halfword Byte-Reverse Indexed
void ac_behavior( lhbrx )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lhbrx r%d, r%d, r%d\n\n",rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
        int ea;

        if (DE_EX.ra!=0)
            ea = ex_value1 + ex_value2;
        else
            ea = ex_value2;

        EX_WB.ea = ea;
        EX_WB.ldwrite = DE_EX.ldwrite;
        EX_WB.lddest = DE_EX.r3;
```



```

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (((int)(MEM.read_byte(EX_WB.ea+1)) & 0x000000FF)<<8) |((int)(MEM.
& 0x000000FF));
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lhz behavior method.
// Load Halfword and Zero
void ac_behavior( lhz )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lhz r%d, %d(r%d)\n\n",rt,d,ra);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
    }

```

```
DE_EX.ldwrite = 1;

} break;

case EX: {
    dbg_printf(" %s \n", get_name());

    int ea;

    if (DE_EX.ra != 0)
        ea = ex_value1 + (short int)DE_EX.d;
    else
        ea = (short int)DE_EX.d;

    EX_WB.ea = ea;
    EX_WB.ldwrite = DE_EX.ldwrite;
    EX_WB.lddest = DE_EX.r3;

} break;

case WB: {
    dbg_printf(" %s \n", get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (unsigned short int)MEM.read_half(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

} break;

case LW: {
    dbg_printf(" %s \n", get_name());
} break;

default: {
} break;
```

```
    }  
  
};  
  
//!Instruction lhzu behavior method.  
// Load Halfword and Zero with Update  
void ac_behavior( lhzu )  
{  
    switch( stage ) {  
        case FE: {  
  
            dbg_printf(" lhzu r%d, %d(%d)\n\n",rt,d,ra);  
  
            } break;  
  
        case DE: {  
  
            DE_EX.regwrite = 1;  
            DE_EX.ldwrite = 1;  
  
            } break;  
  
        case EX: {  
            dbg_printf(" %s \n",get_name());  
  
            int ea = ex.value1 + (short int)DE_EX.d;  
  
            EX_WB.alures = ea;  
            EX_WB.regwrite = DE_EX.regwrite;  
            EX_WB.rdest = DE_EX.ra;  
  
            EX_WB.ea = ea;  
            EX_WB.ldwrite = DE_EX.ldwrite;  
            EX_WB.lddest = DE_EX.r3;
```

```
    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (unsigned short int)MEM.read_half(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lhzux behavior method.
// Load Halfword and Zero with Update Indexed
void ac_behavior( lhzux )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lhzux r%d, r%d, r%d\n\n",rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;
    }
    }
}
```

```
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea = ex_value1 + ex_value2;

    EX_WB.alures = ea;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    EX_WB.ea = ea;
    EX_WB.ldwrite = DE_EX.ldwrite;
    EX_WB.lddest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = (unsigned short int)MEM.read_half(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};
```

```
//!Instruction lhzx behavior method.
// Load Halfword and Zero Indexed
void ac_behavior( lhzx )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lhzx r%d, r%d, r%d\n\n",rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea;

        if (DE.EX.ra!=0)
            ea = ex_value1 + ex_value2;
        else
            ea = ex_value2;

        EX.WB.ea = ea;
        EX.WB.ldwrite = DE.EX.ldwrite;
        EX.WB.lddest = DE.EX.r3;

        } break;

    case WB: {
        dbg_printf(" %s \n",get_name());
```

```
WB_LW.lddest = EX_WB.lddest;
WB_LW.lddata = (unsigned short int)MEM.read_half(EX_WB.ea);
WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lmw behavior method.
// Load Multiple Word
void ac_behavior( lmw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lmw r%d, %d(r%d)\n\n",rt,d,ra);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea;
        unsigned int r;
```

```
if (DE_EX.ra != 0)
    ea = ex_value1 + (short int)DE_EX.d;
else
    ea = (short int)DE_EX.d;

r = DE_EX.r3;

while(r<=31) {
    if((r!=DE_EX.ra)||(r==31))
        GPR[r] = MEM.read(ea);
    r=r+1;
    ea=ea+4;
}

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction lswi behavior method.
// Load String Word Immediate
void ac_behavior( lswi )
{
    switch( stage ) {
```



```
case FE: {

    dbg_printf(" lswi r%d, r%d, %d\n\n",rt,ra,nb);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;
    unsigned int cnt,n;
    unsigned int rfinal,r;
    unsigned int i,masc;

    if (DE_EX.ra!=0)
        ea = ex_value1;
    else
        ea = 0;

    if (DE_EX.nb==0)
        cnt = 32;
    else
        cnt = DE_EX.nb;

    n = cnt;
    rfinal = ((DE_EX.r3 + ceil(cnt,4) - 1) % 32);
    r = DE_EX.r3-1;
    i = 0;

    while(n>0) {
        if(i==0) {
```

```

        r=r+1;
        if(r==32)
            r=0;
        if((r!=ra)||(r==rfinal))
            GPR[r] = 0;
    }
    if((r!=ra)||(r==rfinal)) {
        masc=0xFF000000>>i;
        masc=~masc;
        GPR[r] = (GPR[r] & masc);
        GPR[r] = (((unsigned int)MEM.read_byte(ea)) << (24-i)) |GPR[r];
    }
    i=i+8;
    if(i==32)
        i=0;
    ea=ea+1;
    n=n-1;
}

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

```

```

//!Instruction lswx behavior method.

```

```
// Load String Word Indexed
void ac_behavior( lswx )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" lswx r%d, r%d, r%d\n\n",rt,ra,rb);

            } break;

        case DE: {
            dbg_printf(" %s \n",get_name());
            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            int ea;
            unsigned int cnt,n;
            unsigned int rfinal,r;
            unsigned int i,masc;

            if (DE_EX.ra!=0)
                ea = ex_value1 + ex_value2;
            else
                ea = ex_value2;

            cnt = XER_TBC_read();
            n = cnt;
            rfinal = ((DE_EX.r3 + ceil(cnt,4) - 1) % 32);
            r = DE_EX.r3-1;
            i = 0;

            while (n>0) {
                if (i==0) {
```

```
        r=r+1;
        if(r==32)
            r=0;
        if(((r!=ra) && (r!=rb)) ||(r==rfinal))
            GPR[r] = 0;
    }
    if (((r!=ra) && (r!=rb)) ||(r==rfinal)) {
        masc=0xFF000000>>i;
        masc=~masc;
        GPR[r] = (GPR[r] & masc);
        GPR[r] = (((unsigned int)MEM.read_byte(ea)) << (24-i)) |GPR[r];
    }
    i=i+8;
    if(i==32)
        i=0;
    ea=ea+1;
    n=n-1;
}

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};
```

```
//!Instruction lwbrx behavior method.
// Load Word Byte-Reverse Indexed
void ac_behavior( lwbrx )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" lwbrx r%d, r%d, r%d\n\n",rt,ra,rb);

            } break;

        case DE: {
            dbg_printf(" %s \n",get_name());
            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            int ea;

            if (DE.EX.ra!=0)
                ea = ex_value1 + ex_value2;
            else
                ea = ex_value2;

            EX.WB.ea = ea;
            EX.WB.ldwrite = DE.EX.ldwrite;
            EX.WB.lddest = DE.EX.r3;

            } break;

        case WB: {
            dbg_printf(" %s \n",get_name());

            WB.LW.lddest = EX.WB.lddest;
```

```

WB_LW.lddata = (((unsigned int)MEM.read_byte(EX_WB.ea+3) & 0x000000FF) <<
24) |
    (((unsigned int)MEM.read_byte(EX_WB.ea+2) & 0x000000FF) << 16) |
    (((unsigned int)MEM.read_byte(EX_WB.ea+1) & 0x000000FF) << 8) |
    (((unsigned int)MEM.read_byte(EX_WB.ea) & 0x000000FF));
WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lwz behavior method.
// Load Word and Zero
void ac_behavior( lwz )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lwz r%d, %d(r%d)\n\n",rt,d,ra);

        } break;

    case DE:{

        DE_EX.ldwrite = 1;

        } break;

```

```
case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if (DE_EX.ra!=0)
        ea = ex_value1 + (short int)DE_EX.d;
    else
        ea = (short int)DE_EX.d;

    dbg_printf("ea = %d\n",ea);

    EX_WB.ea = ea;
    EX_WB.ldwrite = DE_EX.ldwrite;
    EX_WB.lddest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = MEM.read(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};
```

```
//!Instruction lwzu behavior method.
// Load Word and Zero with Update
void ac_behavior( lwzu )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" lwzu r%d, %d(r%d)\n\n",rt,d,ra);

            } break;

        case DE:{

            DE_EX.regwrite = 1;
            DE_EX.ldwrite = 1;

            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            int ea = ex_value1 + (short int)DE_EX.d;

            EX_WB.alures = ea;
            EX_WB.regwrite = DE_EX.regwrite;
            EX_WB.rdest = DE_EX.ra;

            EX_WB.ea = ea;
            EX_WB.ldwrite = DE_EX.ldwrite;
            EX_WB.lddest = DE_EX.r3;

            } break;

        case WB: {
```



```
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = MEM.read(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lwzux behavior method.
// Load Word and Zero with Update Indexed
void ac_behavior( lwzux )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lwzux r%d, r%d, r%d\n\n",rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;

        } break;
```

```
case EX: {
    dbg_printf(" %s \n",get_name());

    int ea = ex_value1 + ex_value2;

    EX_WB.alures = ea;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    EX_WB.ea = ea;
    EX_WB.ldwrite = DE_EX.ldwrite;
    EX_WB.lddest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    WB_LW.lddest = EX_WB.lddest;
    WB_LW.lddata = MEM.read(EX_WB.ea);
    WB_LW.ldwrite = EX_WB.ldwrite;

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction lwzx behavior method.
// Load Word and Zero Indexed
```

```
void ac_behavior( lwzx )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" lwzx %d, %d, %d\n\n",rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea;

        if (DE_EX.ra!=0)
            ea = ex_value1 + ex_value2;
        else
            ea = ex_value2;

        EX_WB.ea = ea;
        EX_WB.ldwrite = DE_EX.ldwrite;
        EX_WB.lddest = DE_EX.r3;

        } break;

    case WB: {
        dbg_printf(" %s \n",get_name());

        WB_LW.lddest = EX_WB.lddest;
        WB_LW.lddata = MEM.read(EX_WB.ea);
        WB_LW.ldwrite = EX_WB.ldwrite;
```

```

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

/*****

//!Instruction macchw behavior method.
// Multiply Accumulate Cross Halfword to Word Modulo Signed
void ac_behavior( macchw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" macchw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = genericMac( NORMAL, CROSS, MODULE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
        EX_WB.regwrite = DE_EX.regwrite;

```

```
EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction macchws behavior method.
// Multiply Accumulate Cross Halfword to Word Saturate Signed
void ac_behavior( macchws )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" macchws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o")):(rc==1?".":""))

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
```

```

    EX_WB.alures = genericMac( NORMAL, CROSS, SATURATE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction macchwsu behavior method.
// Multiply Accumulate Cross Halfword to Word Saturate Unsigned
void ac_behavior( macchwsu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" macchwsu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

```

```

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( NORMAL, CROSS, SATURATE, UNSIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction macchwu behavior method.
// Multiply Accumulate Cross Halfword to Word Modulo Unsigned
void ac_behavior( macchwu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" macchwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

        } break;

    case DE: {

```

```

    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( NORMAL, CROSS, MODULE, UNSIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction machhw behavior method.
// Multiply Accumulate High Halfword to Word Modulo Signed
void ac_behavior( machhw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" machhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

```



```

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( NORMAL, HIGH, MODULE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction machhws behavior method.
// Multiply Accumulate High Halfword to Word Saturate Signed
void ac_behavior( machhws )
{
    switch( stage ) {
    case FE: {

```

```

    dbg_printf(" machhws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")))

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = genericMac( NORMAL, HIGH, SATURATE ,SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

//!Instruction machhwsu behavior method.
// Multiply Accumulate High Halfword to Word Saturate Unsigned
void ac_behavior( machhwsu )

```

```
{
switch( stage ) {
case FE: {

    dbg_printf(" machhwsu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"."));

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = genericMac( NORMAL, HIGH, SATURATE, UNSIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};
```

```
//!Instruction machhwu behavior method.
// Multiply Accumulate High Halfword to Word Modulo Unsigned
void ac_behavior( machhwu )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" machhwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

                } break;

        case DE: {
            dbg_printf(" %s \n",get_name());
            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            EX_WB.alures = genericMac( NORMAL, HIGH, MODULE, UNSIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
            EX_WB.regwrite = DE_EX.regwrite;
            EX_WB.rdest = DE_EX.r3;

            } break;

        case WB: {
            dbg_printf(" %s \n",get_name());
            } break;

        case LW: {
            dbg_printf(" %s \n",get_name());
            } break;

        default: {
            } break;
    }
}
```

```
    }  
};  
  
//!Instruction maclhw behavior method.  
// Multiply Accumulate Low Halfword to Word Modulo Signed  
void ac_behavior( maclhw )  
{  
    switch( stage ) {  
        case FE: {  
  
            dbg_printf(" maclhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))  
  
                } break;  
  
        case DE: {  
            dbg_printf(" %s \n", get_name());  
            } break;  
  
        case EX: {  
            dbg_printf(" %s \n", get_name());  
  
            EX_WB.alures = genericMac( NORMAL, LOW, MODULE, SIGNED, oe, rc, ex_value1,  
ex_value2, ex_value3);  
            EX_WB.regwrite = DE_EX.regwrite;  
            EX_WB.rdest = DE_EX.r3;  
  
                } break;  
  
        case WB: {  
            dbg_printf(" %s \n", get_name());  
            } break;  
  
        case LW: {  
            dbg_printf(" %s \n", get_name());  
            } break;  
    }  
}
```

```
    default: {
        } break;
    }
};

//!Instruction maclhws behavior method.
// Multiply Accumulate Low Halfword to Word Saturate Signed
void ac_behavior( maclhws )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" maclhws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":""))

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        EX_WB.alures = genericMac( NORMAL, LOW, SATURATE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        } break;
    case WB: {
        dbg_printf(" %s \n", get_name());
        } break;

    case LW: {
```

```
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction maclhwsu behavior method.
// Multiply Accumulate Low Halfword to Word Saturate Unsigned
void ac_behavior( maclhwsu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" maclhwsu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o")):(rc==1?".":""),

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = genericMac( NORMAL, LOW, SATURATE, UNSIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        } break;

    case WB: {
        dbg_printf(" %s \n",get_name());
```

```
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

//!Instruction maclhwu behavior method.
// Multiply Accumulate Low Halfword to Word Modulo Unsigned
void ac_behavior( maclhwu )
{
    switch( stage ) {
case FE: {

    dbg_printf(" maclhwu%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")))

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = genericMac( NORMAL, LOW, MODULE, UNSIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;
```



```
case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

/*****/

//!Instruction mcrf behavior method.
// Move Condition Register Field
void ac_behavior( mcrf )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mcrf %d, %d\n\n",bf,bfa);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
```

```
unsigned int m = DE_EX.u3b; // bfa
unsigned int n = DE_EX.u3a; // bf
unsigned int tmp, i;
unsigned int masc;
/* n <- m */

/* Clean all bits except m-bits in tmp */
masc = 0xF0000000;
for(i=0 ; i<m ; i++)
    masc=masc>>4;
tmp = CR.read();
tmp = tmp & masc;

/* Put m-bits in n position */
if(n<m)
    for(i=0 ; i < m-n ; i++)
        tmp = tmp << 4;
else
    if(n>m) /* Else nothing */
        for(i=0 ; i < n-m ; i++)
            tmp = tmp >> 4;

/* Clean all bits of n-bits and make an or with tmp */
masc = 0xF0000000;
for(i=0 ; i<n ; i++)
    masc = masc >> 4;
masc = ~masc;
CR.write((CR.read() & masc)|tmp);

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;
```

```
case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction mcrxr behavior method.
// Move to Condition Register from XER
void ac_behavior( mcrxr )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mcrxr %d\n\n",bf);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.u3a = bf;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int n = DE_EX.u3a; // bf
        unsigned int i;
        unsigned int tmp = 0x00;

        /* Calculate tmp bits by XER */
```

```

if(XER_SO_read())
    tmp=tmp |0x80000000;
if(XER_OV_read())
    tmp=tmp |0x40000000;
if(XER_CA_read())
    tmp=tmp |0x20000000;

/* Move altered bits to correct CR field */
for(i=0 ; i<n ; i++)
    tmp=tmp>>4;
CR.write(tmp);

/* Clean XER bits */
XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
XER.write(XER.read() & 0x7FFFFFFF); /* Write 0 to bit 0 SO */
XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction mfcr behavior method.
// Move From Condition Register
void ac_behavior( mfcr )

```

```
{
switch( stage ) {
case FE: {

    dbg_printf(" mfcrr %d\n\n",rt);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = CR.read();
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction mfspr behavior method.
```



```
    case 0x100:
        EX_WB.alures = USPRG0.read();
        break;

    /* SPRG4 */
    case 0x104:
        EX_WB.alures = SPRG4.read();
        break;

    /* SPRG5 */
    case 0x105:
        EX_WB.alures = SPRG5.read();
        break;

    /* SPRG6 */
    case 0x106:
        EX_WB.alures = SPRG6.read();
        break;

    /* SPRG7 */
    case 0x107:
        EX_WB.alures = SPRG7.read();
        break;

    /* Not implemented yet! */
    default:
        dbg_printf("\nERROR!\n");
        exit(-1);
        break;
}

EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.r3;
```

```
    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction mtrcf behavior method.
// Move to Condition Register Fields
void ac_behavior( mtrcf )
{
    switch( stage ) {
case FE: {

    dbg_printf(" mtrcf %d, r%d\n\n",xfm,rs);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    unsigned int tmpop,tmpmask;
```



```
unsigned int mask;
unsigned int i;
unsigned int crm = DE_EX.rf;

tmpmask = 0xF0000000;
tmpop = 0x80;
mask = 0;
for(i=0;i<8;i++) {
    if(crm & tmpop)
        mask=mask |tmpmask;
    tmpmask=tmpmask>>4;
    tmpop=tmpop>>1;
}

CR.write((ex_value1 & mask) |(CR.read() & ~mask));

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

//!Instruction mtspr behavior method.
// Move To Special Purpose Register
```

```
void ac_behavior( mtspr )
{
    switch( stage ) {
    case FE: {

        /* This instruction is a fix, other implementations can be better */
        dbg_printf(" mtspr %d,r%d\n\n",sprf,rs);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int spvalue = DE_EX.rf;
        spvalue=((spvalue>>5) & 0x0000001f ) |
            ((spvalue<<5) & 0x000003e0 );

        switch(spvalue) {

            /* LR */
            case 0x008:
                LR.write(ex_value1);
                dbg_printf(" escreveu %d para lr\n\n",ex_value1);
                break;

            /* CTR */
            case 0x009:
                CTR.write(ex_value1);
                break;

            /* USPRGO */
```

```
    case 0x100:
        USPRG0.write(ex_value1);
        break;

    /* SPRG4 */
    case 0x104:
        SPRG4.write(ex_value1);
        break;

    /* SPRG5 */
    case 0x105:
        SPRG5.write(ex_value1);
        break;

    /* SPRG6 */
    case 0x106:
        SPRG6.write(ex_value1);
        break;

    /* SPRG7 */
    case 0x107:
        SPRG7.write(ex_value1);
        break;

    /* Not implemented yet! */
    default:
        dbg_printf("\nERROR!\n");
        exit(-1);
        break;
}

} break;

case WB: {
    dbg_printf(" %s \n", get_name());
```

```

    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}

};

/*****

//!Instruction mulchw behavior method.
// Multiply Cross Halfword to Word Signed
void ac_behavior( mulchw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mulchw%s r%d, r%d, r%d\n\n", (rc==1?".":""), rt, ra, rb);

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        EX_WB.alures = genericMac( MULTIPLY, CROSS, MODULE, SIGNED, 0, rc, ex_value1,
ex_value2, 0);
        EX_WB.regwrite = DE_EX.regwrite;

```

```
EX_WB.rdest = DE_EX.r3;

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction mulchwu behavior method.
// Multiply Cross Halfword to Word Unsigned
void ac_behavior( mulchwu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mulchwu%s r%d, r%d, r%d\n\n", (rc==1?" ":""),rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
```

```
EX_WB.alures = genericMac( MULTIPLY, CROSS, MODULE, UNSIGNED, 0, rc, ex_value1,
ex_value2, 0);
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.r3;

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction mulhhw behavior method.
// Multiply High Halfword to Word Signed
void ac_behavior( mulhhw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mulhhw%s r%d, r%d, r%d\n\n", (rc==1?" ":""),rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;
    }
```

```

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = genericMac( MULTIPLY, HIGH, MODULE, SIGNED, 0, rc, ex_value1,
ex_value2, 0);
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        } break;

    case WB: {
        dbg_printf(" %s \n",get_name());
        } break;

    case LW: {
        dbg_printf(" %s \n",get_name());
        } break;

    default: {
        } break;
    }
};

//!Instruction mulhhwu behavior method.
// Multiply High Halfword to Word Unsigned
void ac_behavior( mulhhwu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mulhhwu%s r%d, r%d, r%d\n\n", (rc==1?" ":""),rt,ra,rb);

        } break;

    case DE: {

```

```

    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( MULTIPLY, HIGH, MODULE, UNSIGNED, 0, rc, ex_value1,
ex_value2, 0);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

/*****/

//!Instruction mulhw behavior method.
// Multiply High Word
void ac_behavior( mulhw )
{
    switch( stage ) {
    case FE: {

```



```
    dbg_printf(" mulhw%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int high;
    long long int prod =
        (long long int)(int)ex_value1*
        (long long int)(int)ex_value2;

    unsigned long long int shprod=prod;
    shprod = shprod>>32;
    high=shprod;

    EX_WB.alures = high;

    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    if (DE_EX.rc==1)
        CR0_update(high);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
```

```
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction mulhwu behavior method.
// Multiply High Word Unsigned
void ac_behavior( mulhwu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mulhwu%s r%d, r%d, r%d\n\n", (rc==1?".":""),rt,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        unsigned int high;
        unsigned long long int prod =
            (unsigned long long int)(unsigned int)ex_value1*
            (unsigned long long int)(unsigned int)ex_value2;

        prod=prod>>32;
        high=prod;
```

```

EX_WB.alures = high;

EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.r3;

if (DE_EX.rc==1)
    CR0_update(high);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

/*****
//!Instruction mullhw behavior method.
// Multiply Low Halfword to Word Signed
void ac_behavior( mullhw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mullhw%s r%d, r%d, r%d\n\n", (rc==1?" ":""),rt,ra,rb);

        } break;

```

```

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( MULTIPLY, LOW, MODULE, SIGNED, 0, rc, ex_value1,
ex_value2, 0);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction mullhwu behavior method.
// Multiply Low Halfword to Word Unsigned
void ac_behavior( mullhwu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mullhwu%s r%d, r%d, r%d\n\n", (rc==1?" ":""),rt,ra,rb);

```

```
    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( MULTIPLY, LOW, MODULE, UNSIGNED, 0, rc, ex_value1,
ex_value2, 0);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

/*****/

//!Instruction mulli behavior method.
// Multiply Low Immediate
void ac_behavior( mulli )
```

```
{
switch( stage ) {
case FE: {

    dbg_printf(" mulli r%d, r%d, %d\n\n",rt,ra,d);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());

    DE_EX.regwrite = 1;

    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    long long int prod =
        (long long int)(int)ex_value1 * (long long int)(short int)(DE_EX.d);

    int low;
    low=prod;

    EX_WB.alures = low;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
```

```

    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction mullw behavior method.
// Multiply Low Word
void ac_behavior( mullw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" mullw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
        int low;
        long long int prod = (long long int)(int)ex_value1 * (long long int)(int)ex_value

        low=prod;

        EX_WB.alures = low;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        if (DE_EX.oe==1) {

```

```

    if(prod != (long long int)(int)low) {
        XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
        XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
    }
    else
        XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
}

if (DE_EX.rc==1)
    CR0_update(low);
} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction nand behavior method.
// NAND
void ac_behavior( nand )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" nand%s r%d, r%d, r%d\n\n",(rc==1?".":""),ra,rs,rb);

    } break;

```



```
case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = ~(ex_value1 & ex_value2);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}

};

//!Instruction neg behavior method.
// Negate
void ac_behavior( neg )
{
```

```
switch( stage ) {
case FE: {

    dbg_printf(" neg%s r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt,ra);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    long long int longresult = ~((unsigned int)ex_value1)+1;
    int result = ~(ex_value1) + 1;

    if (DE_EX.oe==1) {
        if(longresult != (long long int)result) {
            XER.write(XER.read() |0x40000000); /* Write 1 to bit 1 OV */
            XER.write(XER.read() |0x80000000); /* Write 1 to bit 0 SO */
        }
        else
            XER.write(XER.read() & 0xBFFFFFFF); /* Write 0 to bit 1 OV */
    }

    if (DE_EX.rc==1)
        CR0_update(result);

    EX_WB.alures = result;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;
```

```

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

/***** 405 instructions *****/

//!Instruction nmacchw behavior method.
// Negative Multiply Accumulate Cross Halfword to Word Modulo Signed
void ac_behavior( nmacchw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" nmacchw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        EX_WB.alures = genericMac( NEGATIVE, CROSS, MODULE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);

```

```

EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction nmacchws behavior method.
// Negative Multiply Accumulate Cross Halfword to Word Saturate Signed
void ac_behavior( nmacchws )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" nmacchws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o")):(rc==1?".":"."));

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

```

```

    EX_WB.alures = genericMac( NEGATIVE, CROSS, SATURATE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction nmachhw behavior method.
// Negative Multiplly Accumulate High Halfword to Word Modulo Signed
void ac_behavior( nmachhw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" nmachhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

```

```

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( NEGATIVE, HIGH, MODULE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction nmachws behavior method.
// Negative Multiply Accumulate High Halfword to Word Saturate Signed
void ac_behavior( nmachws )
{
    switch( stage ) {
case FE: {

    dbg_printf(" nmachws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o")):(rc==1?".":""))

    } break;

```

```

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( NEGATIVE, HIGH, SATURATE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction nmaclhw behavior method.
// Negative Multiply Accumulate Low Halfword to Word Modulo Signed
void ac_behavior( nmaclhw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" nmaclhw%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o")):(rc==1?".":""))

```

```
    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = genericMac( NEGATIVE, LOW, MODULE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction nmaclhws behavior method.
// Negative Multiply Accumulate Low Halfword to Word Saturate Signed
void ac_behavior( nmaclhws )
{
    switch( stage ) {
```



```

case FE: {

    dbg_printf(" nmaclhws%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"."));

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = genericMac( NEGATIVE, LOW, SATURATE, SIGNED, oe, rc, ex_value1,
ex_value2, ex_value3);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

```

```

/*****/

```

```
//!Instruction nor behavior method.
// NOR
void ac_behavior( nor )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" nor%s r%d, r%d, r%d\n\n", (rc==1?" ":""),ra,rs,rb);

            } break;

        case DE: {
            dbg_printf(" %s \n",get_name());
            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            EX_WB.alures = ~(ex_value1 |ex_value2);
            EX_WB.regwrite = DE_EX.regwrite;
            EX_WB.rdest = DE_EX.ra;

            if (DE_EX.rc==1)
                CR0_update(EX_WB.alures);

            } break;

        case WB: {
            dbg_printf(" %s \n",get_name());
            } break;

        case LW: {
            dbg_printf(" %s \n",get_name());
            } break;
    }
}
```

```
    default: {
        } break;
    }
};

//!Instruction ore behavior method.
// OR
void ac_behavior( ore )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" or%s r%d, r%d, r%d\n\n", (rc==1?" ":""),ra,rs,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = ex_value1 |ex_value2;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;

        if (DE_EX.rc==1)
            CR0_update(EX_WB.alures);

        } break;

    case WB: {
        dbg_printf(" %s \n",get_name());
        } break;
    }
}
```

```
case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction orc behavior method.
// OR with Complement
void ac_behavior( orc )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" orc%s r%d, r%d, r%d\n\n", (rc==1?" ":""),ra,rs,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = ex_value1 |~ex_value2;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;

        if (DE_EX.rc==1)
            CR0_update(EX_WB.alures);
```

```
    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction ori behavior method.
// OR Immediate
void ac_behavior( ori )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" ori r%d, r%d, %d\n\n",ra,rs,ui);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = ex.value1 |(int)((unsigned short int)DE_EX.ui);
        EX_WB.regwrite = DE_EX.regwrite;
```

```
EX_WB.rdest = DE_EX.ra;

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
};

//!Instruction oris behavior method.
// OR Immediate Shifted
void ac_behavior( oris )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" oris r%d, r%d, r%d\n\n",ra,rs,ui);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
```

```
EX_WB.alures = ex.value1 |((((int)((unsigned short int)DE_EX.ui)) << 16);
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.ra;

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

//!Instruction rlwimi behavior method.
// Rotate Left Word Immediate then Mask Insert
void ac_behavior( rlwimi )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" rlwimi%s r%d, r%d, %d, %d, %d\n\n", (rc==1?" ":""),ra,rs,sh,mb,me);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
```

```

    dbg_printf(" %s \n",get_name());
    unsigned int r = rotl(ex_value2, DE_EX.sh);
    unsigned int m = mask32rlw(DE_EX.mb, DE_EX.me);

    EX_WB.alures = (r & m) |(ex_value1 & ~m);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}

};

//!Instruction rlwinm behavior method.
// Rotate Left Word Immediate then AND with Mask
void ac_behavior( rlwinm )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" rlwinm%s r%d, r%d, %d, %d, %d\n\n", (rc==1?" ":""),ra,rs,sh,mb,me);
    }
    }
}

```



```
    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    unsigned int r = rotl(ex.value2, DE_EX.sh);
    unsigned int m = mask32rlw(DE_EX.mb, DE_EX.me);

    EX_WB.alures = (r & m);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    if (DE_EX.rc==1)
        CRO_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}

};
```

```
//!Instruction rlwnm behavior method.
// Rotate Left Word then AND with Mask
void ac_behavior( rlwnm )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" rlwnm%s r%d, r%d, %d, %d, %d\n\n", (rc==1?".":""), ra, rs, rb, mb, me);

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        unsigned int r = rotl(ex_value1, (ex_value2 | 0x0000001F));
        unsigned int m = mask32rlw(DE_EX.mb, DE_EX.me);

        EX_WB.alures = (r & m);
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;

        if (DE_EX.rc==1)
            CR0_update(EX_WB.alures);

        } break;

    case WB: {
        dbg_printf(" %s \n", get_name());
        } break;
    }
```

```
case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction sc behavior method.
// System Call
/* The registers used in this intruction may be defined better */
void ac_behavior( sc )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" sc\n\n");

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.npc = FE.DE.npc;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        SRR1.write(MSR.read()); /* It uses a function to join MSR fields */

        /* Write WE, EE, PR, DR and IR as 0 in MSR */
        MSR.write(MSR.read() |0xFFB3FCF);
```

```
SRRO.write(DE_EX.npc); /* Only to understand pre-increment */

ac_pc = ((EVPR.read() & 0xFFFF0000) | 0x0000C00);
ac_flush("FE");
ac_flush("DE");

} break;

case WB: {
    dbg_printf(" %s \n", get_name());
} break;

case LW: {
    dbg_printf(" %s \n", get_name());
} break;

default: {
} break;
}

};

//!Instruction slw behavior method.
// Shift Left Word
void ac_behavior( slw )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" slw%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);

            } break;

        case DE: {
```

```
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    unsigned int n=(ex_value2 & 0x0000001F);
    unsigned int r=rotr(ex_value1,n);
    unsigned int m;

    if((ex_value2 & 0x00000020)==0x00) /* Check bit 26 */
        m=mask32rlw(0,31-n);
    else
        m=0;

    EX_WB.alures = r & m;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
```

```
};

//!Instruction sraw behavior method.
// Shift Right Algebraic Word
void ac_behavior( sraw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" sraw%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        unsigned int n=(ex_value2 & 0x0000001F);
        unsigned int r=rotr(ex_value1, 32-n);
        unsigned int m;
        unsigned int s;

        if((ex_value2 & 0x00000020)==0x00) /* Check bit 26 */
            m=mask32rlw(n, 31);
        else
            m=0;

        s=(ex_value1 & 0x80000000);
        if(s!=0)
            s=0xFFFFFFFF;
    }
}
```

```

/* Write ra */
EX_WB.alures = (r & m) |(s & ~m);
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.ra;

/* Update XER CA */
if(s && ((r & ~m)!=0))
    XER.write(XER.read() |0x20000000); /* Write 1 to bit 2 CA */
else
    XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */

/* Update CR register */
if (DE_EX.rc==1)
    CR0_update(EX_WB.alures);

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}

};

//!Instruction srawi behavior method.
// Shift Right Algebraic Word Immediate
void ac_behavior( srawi )
{

```

```
switch( stage ) {
case FE: {

    dbg_printf(" srawi%s r%d, r%d, %d\n\n", (rc==1?" ":""), ra, rs, sh);

    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    unsigned int n = DE_EX.rb; // sh
    unsigned int r = rotl(ex_value1, 32-n);
    unsigned int m = mask32rlw(n, 31);
    unsigned int s = (ex_value1 & 0x80000000);

    if(s!=0)
        s=0xFFFFFFFF;

    int result=(r & m) |(s & ~m);

    EX_WB.alures = result;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    /* Update XER CA */
    if(s && ((r & ~m)!=0))
        XER.write(XER.read() |0x20000000); /* Write 1 to bit 2 CA */
    else
        XER.write(XER.read() & 0xDFFFFFFF); /* Write 0 to bit 2 CA */

    if (DE_EX.rc==1)
```



```
        CR0_update(result);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction srw behavior method.
// Shift Right Word
void ac_behavior( srw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" srw%s r%d, r%d, r%d\n\n", (rc==1?" ":""),ra,rs,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
```

```
unsigned int n=(ex_value2 & 0x0000001F);
unsigned int r=rotr(ex_value1,32-n);
unsigned int m;

if((ex_value2 & 0x00000020)==0x00) /* Check bit 26 */
    m=mask32rlw(n,31);
else
    m=0;

EX_WB.alures = r & m;
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.ra;

if (DE_EX.rc==1)
    CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction stb behavior method.
// Store Byte
void ac_behavior( stb )
{
```

```
switch( stage ) {
case FE: {

    dbg_printf(" stb r%d, %d(r%d)\n\n",rs,d,ra);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if (DE_EX.ra!=0)
        ea = ex.value1 + (short int)DE_EX.d;
    else
        ea = (short int)DE_EX.d;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex.value2;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_byte(EX_WB.ea,(unsigned char)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
char)EX_WB.wdata);

    } break;
```

```
case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction stbu behavior method.
// Store Byte with Update
void ac_behavior( stbu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" stbu r%d, %d(r%d)\n\n",rs,d,ra);

        } break;

    case DE:{

        DE_EX.regwrite = 1;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea = ex_value1 + (short int)DE_EX.d;

        EX_WB.alures = ea;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;
```

```
EX_WB.ea = ea;
EX_WB.stwrite = DE_EX.stwrite;
EX_WB.wdata = ex_value2;

} break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_byte(EX_WB.ea,(unsigned char)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
char)EX_WB.wdata);

} break;

case LW: {
    dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

//!Instruction stbux behavior method.
// Store Byte with Update Indexed
void ac_behavior( stbux )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" stbux r%d, r%d, r%d\n\n",rs,ra,rb);

    } break;
```

```
case DE: {
    dbg_printf(" %s \n",get_name());

    DE_EX.regwrite = 1;

    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea = ex_value1 + ex_value2;

    EX_WB.alures = ea;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex_value3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_byte(EX_WB.ea,(unsigned char)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
char)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;
```

```
default: {
    } break;
}
};

//!Instruction stbx behavior method.
// Store Byte Indexed
void ac_behavior( stbx )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" stbx r%d, r%d, r%d\n\n",rs,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea;

        if (DE_EX.ra!=0)
            ea = ex_value1 + ex_value2;
        else
            ea = ex_value2;

        EX_WB.ea = ea;
        EX_WB.stwrite = DE_EX.stwrite;
        EX_WB.wdata = ex_value3;
```

```
    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_byte(EX_WB.ea,(unsigned char)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
char)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction sth behavior method.
// Store Halfword
void ac_behavior( sth )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" sth r%d, %d(r%d)\n\n",rs,d,ra);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;
    }
```



```
case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if (DE_EX.ra!=0)
        ea = ex_value1 + (short int)DE_EX.d;
    else
        ea = (short int)DE_EX.d;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex_value2;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_half(EX_WB.ea,(unsigned short int)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
short int)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction sthbrx behavior method.
```

```
// Store Halfword Byte-Reverse Indexed
void ac_behavior( sthbrx )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" sthbrx r%d, r%d, r%d\n\n",rs,ra,rb);

            } break;

        case DE: {
            dbg_printf(" %s \n",get_name());
            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            int ea;

            if (DE.EX.ra!=0)
                ea = ex_value1 + ex_value2;
            else
                ea = ex_value2;

            EX_WB.ea = ea;
            EX_WB.stwrite = DE.EX.stwrite;
            EX_WB.wdata = ex_value3;

            } break;

        case WB: {
            dbg_printf(" %s \n",get_name());

            MEM.write_half(EX_WB.ea,(unsigned short int)
                ( ((EX_WB.wdata & 0x000000FF) << 8) |
```

```

        ((EX_WB.wdata & 0x0000FF00) >> 8) ));
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
short int)
        ( ((EX_WB.wdata & 0x000000FF) << 8) |
          ((EX_WB.wdata & 0x0000FF00) >> 8) ));

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
}

//!Instruction sthu behavior method.
// Store Halfword with Update
void ac_behavior( sthu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" sthu r%d, %d(r%d)\n\n",rs,d,ra);

        } break;

    case DE:{

        DE_EX.regwrite = 1;

        } break;

    case EX: {

```

```
    dbg_printf(" %s \n",get_name());

    int ea = ex_value1 + (short int)DE_EX.d;

    EX_WB.alures = ea;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex_value2;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_half(EX_WB.ea,(unsigned short int)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
short int)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction sthux behavior method.
// Store Halfword with Update Indexed
void ac_behavior( sthux )
```

```
{
switch( stage ) {
case FE: {

    dbg_printf("sthux r%d, r%d, r%d\n\n",rs,ra,rb);

    } break;

case DE:{

    DE_EX.regwrite = 1;

    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea = ex.value1 + ex.value2;

    EX_WB.alures = ea;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex.value3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_half(EX_WB.ea,(unsigned short int)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
short int)EX_WB.wdata);
```

```
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction sthx behavior method.
// Store Halfword Indexed
void ac_behavior( sthx )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" sthx r%d, r%d, r%d\n\n",rs,ra,rb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea;

        if (ra!=0)
            ea = ex.value1 + ex.value2;
        else
```

```
    ea = ex.value2;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex.value3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write_half(EX_WB.ea,(unsigned short int)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
short int)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction stmw behavior method.
// Store Multiple Word
void ac_behavior( stmw )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" stmw r%d, %d(r%d)\n\n",rs,d,ra);
```

```
    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    int ea;
    unsigned int r;

    if (DE_EX.ra!=0)
        ea = ex_value1 + (short int)DE_EX.d;
    else
        ea = (short int)DE_EX.d;

    r = DE_EX.r3;

    while(r<=31) {
        MEM.write(ea, GPR[r]);
        dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)ea, (ac_Uword)GPR[r]);
        r+=1;
        ea+=4;
    }

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;
```



```
    default: {
        } break;
    }
};

//!Instruction stswi behavior method.
// Store String Word Immediate
void ac_behavior( stswi )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" stswi r%d, r%d, %d\n\n",rs,ra,nb);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea;
        unsigned int n;
        unsigned int r;
        unsigned int i,masc;

        if (DE_EX.ra!=0)
            ea = ex.value1;
        else
            ea = 0;

        if (DE_EX.nb==0)
```

```
    n = 32;
else
    n = DE_EX.nb;

r = DE_EX.r3-1;
i = 0;

while(n>0) {
    if(i==0)
        r=r+1;
    if(r==32)
        r=0;
    masc=mask32rlw(i,i+7);
    MEM.write_byte(ea,(unsigned char)((GPR[r] & masc) >> (24-i)));
    dbg_printf("\n== writing MEM[%#x]=%#x \n", ea, (unsigned char)((GPR[r]
& masc) >> (24-i)));
    i=i+8;
    if(i==32)
        i=0;
    ea=ea+1;
    n=n-1;
}

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
```

```
    }  
};  
  
//!Instruction stswx behavior method.  
// Store String Word Indexed  
void ac_behavior( stswx )  
{  
    switch( stage ) {  
    case FE: {  
  
        dbg_printf(" stswx r%d, r%d, r%d\n\n",rs,ra,rb);  
  
        } break;  
  
    case DE: {  
        dbg_printf(" %s \n",get_name());  
        } break;  
  
    case EX: {  
        dbg_printf(" %s \n",get_name());  
  
        int ea;  
        unsigned int n;  
        unsigned int r;  
        unsigned int i,masc;  
  
        if (DE_EX.ra!=0)  
            ea = ex_value1 + ex_value2;  
        else  
            ea = ex_value2;  
  
        n = XER_TBC_read();  
  
        r = DE_EX.r3-1;  
        i = 0;
```

```
while(n>0) {
    if(i==0)
        r=r+1;
    if(r==32)
        r=0;
    masc=mask32rlw(i,i+7);
    MEM.write_byte(ea,(unsigned char)((GPR[r] & masc) >> (24-i)));
    dbg_printf("\n== writing MEM[%#x]=%#x \n", ea, (unsigned char)((GPR[r]
& masc) >> (24-i)));
    i=i+8;
    if(i==32)
        i=0;
    ea=ea+1;
    n=n-1;
}

} break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction stw behavior method.
// Store Word
void ac_behavior( stw )
```

```
{
switch( stage ) {
case FE: {

    dbg_printf(" stw r%d, %d(r%d)\n\n",rs,d,ra);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if (DE_EX.ra!=0)
        ea = ex_value1 + (short int)DE_EX.d;
    else
        ea = (short int)DE_EX.d;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex_value2;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write(EX_WB.ea,(unsigned int)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
int)EX_WB.wdata);
```

```
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction stwbrx behavior method.
// Store Word Byte-Reverse Indexed
void ac_behavior( stwbrx )
{
    switch( stage ) {
case FE: {

    dbg_printf(" stwbrx r%d, r%d, r%d\n\n",rs,ra,rb);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if (DE_EX.ra!=0)
        ea = ex_value1 + ex_value2;
    else
        ea = ex_value2;
```

```

EX_WB.ea = ea;
EX_WB.stwrite = DE_EX.stwrite;
EX_WB.wdata = ex_value3;

} break;

case WB: {
  dbg_printf(" %s \n",get_name());

  MEM.write(EX_WB.ea,
    (((EX_WB.wdata & 0x000000FF) << 24) |
    ((EX_WB.wdata & 0x0000FF00) << 8 ) |
    ((EX_WB.wdata & 0x00FF0000) >> 8 ) |
    ((EX_WB.wdata & 0xFF000000) >> 24)));
  dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea,
    (((EX_WB.wdata & 0x000000FF) << 24) |
    ((EX_WB.wdata & 0x0000FF00) << 8 ) |
    ((EX_WB.wdata & 0x00FF0000) >> 8 ) |
    ((EX_WB.wdata & 0xFF000000) >> 24)));

} break;

case LW: {
  dbg_printf(" %s \n",get_name());
} break;

default: {
} break;
}
};

//!Instruction stwu behavior method.
// Store Word with Update

```

```
void ac_behavior( stwu )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" stwu r%d, %d(r%d)\n\n",rs,d,ra);

        } break;

    case DE:{
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea = ex_value1 + (short int)DE_EX.d;

        EX_WB.alures = ea;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.ra;

        EX_WB.ea = ea;
        EX_WB.stwrite = DE_EX.stwrite;
        EX_WB.wdata = ex_value2;

        } break;

    case WB: {
        dbg_printf(" %s \n",get_name());

        MEM.write(EX_WB.ea,(unsigned int)EX_WB.wdata);
```



```
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
int)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction stwux behavior method.
// Store Word with Update Indexed
void ac_behavior( stwux )
{
    switch( stage ) {
    case FE: {

        dbg_printf("stwux r%d, r%d, r%d\n\n",rs,ra,rb);

        } break;

    case DE:{

        DE_EX.regwrite = 1;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ea = ex_value1 + ex_value2;
```

```

EX_WB.alures = ea;
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.ra;

EX_WB.ea = ea;
EX_WB.stwrite = DE_EX.stwrite;
EX_WB.wdata = ex_value3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write(EX_WB.ea,EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (ac_Uword)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction stwx behavior method.
// Store Word Indexed
void ac_behavior( stwx )
{
    switch( stage ) {
    case FE: {

```

```
    dbg_printf(" stwx r%d, r%d, r%d\n\n",rs,ra,rb);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    int ea;

    if(ra!=0)
        ea = ex_value1 + ex_value2;
    else
        ea = ex_value2;

    EX_WB.ea = ea;
    EX_WB.stwrite = DE_EX.stwrite;
    EX_WB.wdata = ex_value3;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());

    MEM.write(EX_WB.ea,(unsigned int)EX_WB.wdata);
    dbg_printf("\n== writing MEM[%#x]=%#x \n", (ac_Uword)EX_WB.ea, (unsigned
int)EX_WB.wdata);

    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
```

```
    } break;

default: {
    } break;
}
};

//!Instruction subf behavior method.
// Subtract From
void ac_behavior( subf )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" subf%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")), r

        } break;

    case DE: {
        dbg_printf(" %s \n", get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n", get_name());

        EX_WB.alures = ~ex_value1 + ex_value2 + 1;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        if (DE_EX.oe==1)
            add_XER_OV_SO_update(EX_WB.alures, ~ex_value1, ex_value2, 1);
        if (DE_EX.rc==1)
            CR0_update(EX_WB.alures);

        } break;
    }
```

```
case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction subfc behavior method.
// Subtract From Carrying
void ac_behavior( subfc )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" subfc%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = ~ex_value1 + ex_value2 + 1;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;
```

```

add_XER_CA_update(EX.WB.alures, ~ex_value1, ex_value2, 1);

if (DE_EX.oe==1)
    add_XER_OV_SO_update(EX.WB.alures, ~ex_value1, ex_value2, 1);
if (DE_EX.rc==1)
    CR0_update(EX.WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

//!Instruction subfe behavior method.
// Subtract From Extended
void ac_behavior( subfe )
{

switch( stage ) {
case FE: {

    dbg_printf(" subfe%s r%d, r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),

    } break;

```

```
case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = ~ex_value1 + ex_value2 + XER_CA_read();
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.r3;

    add_XER_CA_update(EX_WB.alures,~ex_value1,ex_value2,XER_CA_read());

    if (DE_EX.oe==1)
        add_XER_OV_SO_update(EX_WB.alures,~ex_value1,ex_value2,XER_CA_read());
    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction subfic behavior method.
// Subtract From Immediate Carrying
```

```
void ac_behavior( subfic )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" subfic r%d, r%d, %d\n\n",rt,ra,d);

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());

        DE_EX.regwrite = 1;

        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        int ime32 = DE_EX.d;
        int result = ~ex_value1 + ime32 + 1;

        EX_WB.alures = result;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        add_XER_CA_update(result, ~ex_value1, ime32, 1);

        } break;

    case WB: {
        dbg_printf(" %s \n",get_name());
        } break;

    case LW: {
```



```

    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction subfme behavior method.
// Subtract from Minus One Extended
void ac_behavior( subfme )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" subfme%s r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt,r

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());

        EX_WB.alures = ~ex_value1 + XER_CA_read() + (-1);
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        add_XER_CA_update(EX_WB.alures, ~ex_value1, XER_CA_read(), -1);

        if (DE_EX.oe==1)
            add_XER_OV_SO_update(EX_WB.alures, ~ex_value1, XER_CA_read(), -1);
        if (DE_EX.rc==1)

```

```
    CR0_update(EX.WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction subfze behavior method.
// Subtract from Zero Extended
void ac_behavior( subfze )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" subfze%s r%d, r%d\n\n", (oe==1?(rc==1?"o.":"o"):(rc==1?".":"")),rt,r

        } break;

    case DE: {
        dbg_printf(" %s \n",get_name());
        } break;

    case EX: {
        dbg_printf(" %s \n",get_name());
```

```
EX_WB.alures = ~ex_value1 + XER_CA_read();
EX_WB.regwrite = DE_EX.regwrite;
EX_WB.rdest = DE_EX.r3;

add_XER_CA_update(EX_WB.alures, ~ex_value1, XER_CA_read(), 0);

if (DE_EX.oe==1)
    add_XER_OV_SO_update(EX_WB.alures, ~ex_value1, XER_CA_read(), 0);
if (DE_EX.rc==1)
    CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}
};

//!Instruction xor behavior method.
// XOR
void ac_behavior( xxor )
{
    switch( stage ) {
    case FE: {

        dbg_printf(" xor%s r%d, r%d, r%d\n\n", (rc==1?" ":""), ra, rs, rb);
```

```
    } break;

case DE: {
    dbg_printf(" %s \n", get_name());
    } break;

case EX: {
    dbg_printf(" %s \n", get_name());

    EX_WB.alures = ex.value1 ^ ex.value2;
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    if (DE_EX.rc==1)
        CR0_update(EX_WB.alures);

    } break;

case WB: {
    dbg_printf(" %s \n", get_name());
    } break;

case LW: {
    dbg_printf(" %s \n", get_name());
    } break;

default: {
    } break;
}

};

//!Instruction xori behavior method.
// XOR immediate
void ac_behavior( xori )
```

```
{
switch( stage ) {
case FE: {

    dbg_printf(" xori r%d, r%d, %d\n\n",ra,rs,ui);

    } break;

case DE: {
    dbg_printf(" %s \n",get_name());
    } break;

case EX: {
    dbg_printf(" %s \n",get_name());

    EX_WB.alures = ex_value1 ^ (int)((unsigned short int)DE_EX.ui);
    EX_WB.regwrite = DE_EX.regwrite;
    EX_WB.rdest = DE_EX.ra;

    } break;

case WB: {
    dbg_printf(" %s \n",get_name());
    } break;

case LW: {
    dbg_printf(" %s \n",get_name());
    } break;

default: {
    } break;
}
};

//!Instruction xoris behavior method.
```

```
// XOR immediate shifted
void ac_behavior( xoris )
{
    switch( stage ) {
        case FE: {

            dbg_printf(" xoris r%d, r%d, %d\n\n",ra,rs,ui);

            } break;

        case DE: {
            dbg_printf(" %s \n",get_name());
            } break;

        case EX: {
            dbg_printf(" %s \n",get_name());

            EX_WB.alures = ex.value1 ^(((int)((unsigned short int)DE_EX.ui)) << 16);
            EX_WB.regwrite = DE_EX.regwrite;
            EX_WB.rdest = DE_EX.ra;

            } break;

        case WB: {
            dbg_printf(" %s \n",get_name());
            } break;

        case LW: {
            dbg_printf(" %s \n",get_name());
            } break;

        default: {
            } break;
        }
};
```


Anexo B - Artigo

MODELAGEM DO PROCESSADOR POWERPC 405 PARA PROJETO BASEADO EM PLATAFORMA

Sandro Rogério Silva de Carvalho

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina
Florianópolis, SC, Brasil

sandroc@inf.ufsc.br

RESUMO

A exploração de alternativas de projeto de sistemas computacionais embarcados requer o uso de modelos de processadores voltados à simulação, visando obter estimativas de consumo de memória, tempo de processamento e energia, que são variáveis importantes neste contexto. Este artigo descreve um modelo do processador PowerPC 405 em dois níveis de abstração: funcional e com precisão de ciclos. Para tal, utilizou-se a Linguagem de Descrição de Arquiteturas (ADL) ArchC. O modelo possibilita a obtenção de simuladores capazes de executar o mesmo binário utilizado no processador real e pode ser integrado na Plataforma de Referência ArchC (ARP) para a comunicação com outros módulos de sistema. A validação se deu através da execução simulada de *benchmarks* de diversos nichos do mercado de sistemas embarcados.

1. INTRODUÇÃO

O aumento do número de transistores que podem ser fabricados em uma dada área de silício vem aumentando conforme a previsão empírica que Gordon Moore propôs em 1965, dizendo ser possível dobrar esse número a cada vinte e quatro meses [1]. Isto aliado à excessiva demanda por aplicações cada vez mais complexas da indústria de sistemas embarcados impulsionou o desenvolvimento de sistemas inteiros em um único chip, os *System-on-Chip* (SoCs). Esses sistemas envolvem a combinação de *hardware* (processadores, memórias e periféricos) e *software* (*Hardware Dependable Software* e aplicação) [5].

A utilização de plataformas, biblioteca de componentes de *hardware* e *software*, possibilita uma redução nos custos envolvidos e no tempo em que o produto chega ao mercado.

A descrição das plataformas é feita utilizando-se Linguagens de Descrição de Sistemas (SDL - *System Description Language*), como é o caso de SystemC [7], que pode ser utilizada em diversos níveis de descrição,

O PowerPC405 é a terceira mais vendida arquitetura de instruções [2], bastante utilizado em sistemas embarcados com a função de controle das operações do sistema ou como unidade principal de processamento. Este processador é um dos muitos fabricados ou licenciados para fabricação pela IBM. É também utilizado como unidade de processamento central (core) de sistemas embarcados a serem inseridos em chips de FPGA da Xilinx.

Neste artigo, a Seção 2 apresenta trabalhos correlatos; a Seção 3 descreve os modelos funcional e com precisão de ciclos do PowerPC 405 e algumas de suas características; a Seção 4 apresenta os resultados experimentais, a Seção 5 conclui o trabalho e vislumbra possíveis trabalhos futuros e, por fim, a Seção 6 apresenta os agradecimentos.

2. TRABALHOS CORRELATOS

SystemC [7] estende a linguagem C++ e permite a descrição de sistemas computacionais desde o nível comportamental até o RTL.

A implementação de um processador em SystemC pode se tornar proibitiva devido à extensa quantidade de estruturas presentes em tal módulo de *hardware*.

Sendo assim, as ADL's abstraem pormenores estruturais do hardware, deixando o projetista livre para se concentrar no comportamento e no formato das instruções.

Aqui utilizamos a ADL ArchC, desenvolvida pelo *Computer Systems Laboratory* (LSC) da Universidade de Campinas (Unicamp), a qual possibilita gerar simuladores interpretados e compilados, montadores, ligadores, desmontadores e depuradores. Esses dois últimos foram

desenvolvidos no Laboratório de Automação de Projeto de Sistemas(LAPS) da Universidade Federal de Santa Catarina (UFSC), do qual o autor faz parte.

Alguns modelos já estão disponíveis em ArchC: PowerPC, MIPS-I, SPARC-V8, Intel 8051 e PIC16F84. Sendo que todos possuem um modelo funcional e alguns o modelo com precisão de ciclos.

O modelo aqui descrito baseou-se em dois modelos funcionais do PowerPC, um que está disponível no site do projeto ArchC e outro que foi objeto de um estudo de caso do trabalho de conclusão de curso de dois membros do LAPS.

A extensão do conjunto de instruções possibilita a utilização de instruções MAC, visando uma exploração de todo o potencial computacional do PowerPC 405. A descrição da versão com precisão de ciclos provê estimativa de tempo de processamento da aplicação a que se destina o sistema.

3. DESCRIÇÃO DOS MODELOS

O PowerPC405 é um processador RISC *load-store* 32 bits largamente utilizado em sistemas embarcados. Ele é uma implementação da arquitetura PowerPC para ambiente-embutido. Garante alta performance e baixo consumo de energia às aplicações embutidas de ponto-fixa.

Pode ser utilizado tanto como uma máquina *big-endian* quanto *little-endian*. Faz uso de uma Unidade de Multiplicação-Soma (*MAC - Multiply-Accumulate Unit*) e não possui unidade de ponto-flutuante. [3] [4]

3.1. Implementação do modelo funcional

Composto por cinco arquivos:

- ppc405.ac
- ppc405_isa.ac
- ppc405-isa.cpp
- ppc405_syscall.cpp
- ppc405_gdb_funcs.cpp

Partimos da descrição estrutural da arquitetura no arquivo ppc405.ac, conforme mostra a figura 1. Podemos notar a definição do tamanho da palavra do processador (*ac_wordsize*), da quantidade de memória disponível e nome que a referencia (*ac_mem*), quantidade de registradores de propósito geral e o nome do banco (*ac_regbank*), a declaração de outros registradores (*ac_reg*), a indicação do arquivo que contém a declaração das instruções (*ac_isa*) e, por fim, a orientação dos bytes nas palavras (*ac_endian*).

No segundo arquivo, ppc405_isa.ac, cujo fragmento está na figura 2, descrevemos como as instruções são

codificadas. Isto é feito da seguinte forma: declaramos os formatos de instruções (*ac_format*), detalhando seus campos, e declaramos as instruções indicando seu formato (*ac_instr*).

Dentro do construtor indicamos as formas alternativas de escrever uma instrução que influenciam no valor de seus campos, pois aqueles que não estiverem explícito no assembly (*set_asm*), são indicados como um campo constante da instrução (*ac_decoder*).

```
AC_ARCH(ppc405) {
    ac_wordsize 32;

    ac_mem MEM:8M;

    ac_regbank GPR:32;

    ac_reg SPRG4;
    ac_reg SPRG5;
    ac_reg SPRG6;
    ac_reg SPRG7;
    ac_reg USPRG0;

    ac_reg XER;

    ac_reg MSR;

    ac_reg EVPR;
    ac_reg SRR0;
    ac_reg SRR1;

    ac_reg CR;
    ac_reg LR;
    ac_reg CTR;

    ARCH_CTOR(ppc405) {
        ac_isa("ppc405_isa.ac");
        set_endian("big");
    };
};
```

Figura 1 - Descrição dos parâmetros do processador no modelo funcional.

```
AC_ISA(ppc405) {
    ac_format I1 = "%opcd:6 %li:24:s %aa:1 %lk:1";
    ac_format X01 = "%opcd:6 %rt:5 %ra:5 %rb:5 %oe:1 %xos:9 %rc:1";
    ...

    ac_instr<I1> b;
    ac_instr<X01> add;
    ...

    ISA_CTOR(ppc405) {
        add.set_asm("add %reg, %reg, %reg", rt, ra, rb, oe=0, rc=0);
        add.set_asm("add. %reg, %reg, %reg", rt, ra, rb, oe=0, rc=1);
        add.set_asm("addo %reg, %reg, %reg", rt, ra, rb, oe=1, rc=0);
        add.set_asm("addo. %reg, %reg, %reg", rt, ra, rb, oe=1, rc=1);
        add.set_decoder(opcd=31, xos=266);

        b.set_asm("b %addrRAu", li, aa=0, lk=0);
        b.set_asm("ba %addrRAu", li, aa=1, lk=0);
        b.set_asm("bl %addrRAu", li, aa=0, lk=1);
        b.set_asm("bla %addrRAu", li, aa=1, lk=1);
        b.set_decoder(opcd=18);
        ...

        pseudo_instr("mr %reg, %reg") {
            "or %0, %1, %1";
        }
        ...
    };
};
```

Figura 2 - Informações de codificação das instruções.

No exemplo da figura, a instrução `add` tem o formato `XOI` e possui quatro formas alternativas de ser escrita:

- `add` - Somente soma;
- `add.` - Soma com atualização do registrador de condição;
- `addo` - Soma com indicação de overflow;
- `addo.` - Soma com atualização do registrador de condição e indicação de overflow.

Note que a instrução `add` é única. Porém a adição dos sufixos citados alteram seus campos `RC` e `OE`, influenciando seu fluxo de execução.

Também no construtor há a indicação de pseudo-instruções, ou seja, aquelas que são válidas na linguagem de montagem, porém não existem no processador. Como é o caso da instrução `mr`.

No arquivo `ppc405-isa.cpp` descrevemos o comportamento de cada instrução. Aqui pode-se usar todos os recursos que a linguagem `SystemC` oferece.

Há uma hierarquia de comportamentos que segue a seguinte regra: primeiro é executado o comportamento `instruction`, comum a todas as instruções, depois o comportamento referente ao formato da instrução, e só então é executado o comportamento específico da instrução. Isto evita a reescrita da parte do código que é comum a todas as instruções ou a um determinado formato.

Na figura 3 há dois exemplos de descrição de comportamento. A instrução `add`, já comentada anteriormente, soma o conteúdo dos registradores-fonte e grava no registrador de destino. Os campos `RC` e `OE` indicam se as etapas alternativas serão executadas.

Já a instrução `macchw` foi colocada aí para ilustrar como foram implementadas as instruções da unidade `MAC`. Elas chamam uma função que tem seu fluxo de execução ditado pelos parâmetros passados. Isso evitou clonagem de código e facilitou na depuração das instruções.

O arquivo `ppc405_syscall.cpp` contém a implementação de especializações dos métodos da classe `ac_syscall` que foram feitos para permitir suporte a chamadas de sistema operacional às aplicações, utilizadas, por exemplo, para escrever e ler arquivos no computador onde está sendo executado o simulador.

O arquivo `ppc405_gdb_funcs.cpp` implementa funções necessárias para se utilizar o modelo com o depurador `gdb`, permitindo escrever e ler registradores e posições de memória.

3.2. Implementação do modelo com precisão de ciclos

```

//!Instruction add behavior method.
// Add
void ac_behavior( add )
{
    int result = GPR[ra] + GPR[rb];

    if (oe==1)
        add_XBR_OV_SO_update(result,GPR[ra],GPR[rb],0);

    if (rc==1)
        CRO_update(result);

    GPR[rt] = result;
};

//!Instruction macchw behavior method.
// Multiply Accumulate Cross Halfword to Word Modulo Signed
void ac_behavior( macchw )
{
    genericMac ( NORMAL , CROSS , MODULE , SIGNED , oe , rc , rt , ra , rb );
};

```

Figura 3 - Descrição do comportamento das instruções `add` e `macchw` no modelo funcional.

O modelo do `PowerPC405` com precisão de ciclo fofoiteo baseado no modelo puramente funcional descrito na seção anterior. As mudanças necessárias para transformar aquele modelo em um que refletisse o tempo necessário para executar a aplicação foram a adição de informações referentes ao pipeline e a descrição do comportamento dividido entre os diferentes estágios do `pipeline`.

Na figura 4 vemos o fragmento de código adicionado ao arquivo `ppc405.ac` que deu origem ao arquivo `ppc405_ca.ac`. Note a declaração dos registradores necessários para o transporte de dados entre os estágios (`ac_reg`), cada um com o seu formato específico (`ac_format`). Há também a declaração do nome dos estágios e sua sequência de execução (`ac_stage`).

```

ac_format F_FR_DE = "%npc:32";
ac_format F_DE_EX = "%npc:32 %data1:32 %data2:32 %data3:32 %ra:5 %rb:5 \
%r3:5 %mb:5 %me:5 %sh:5 %mb:5 %u3a:3 %u3b:3 %rf:10 \
%regwrite:1 %oe:1 %rc:1 %ui:16 %si:16 %s %d:16:s \
%ldwrite:1 %stwrite:1 %bo:5 %bi:5 %bd:14:s %aa:1 \
%lk:1 %li:24:s";
ac_format F_EX_WB = "%alures:32 %wdata:32 %ea:32 %rdest:5 %regwrite:1 \
%stwrite:1 %ldwrite:1 %lddest:5";
ac_format F_WB_LW = "%lddata:32 %lddest:5 %ldwrite:1";

/* FETch, DEcode, EXecute, Write-Back and Load Write-back */
ac_reg<F_FR_DE> FR_DE;
ac_reg<F_DE_EX> DE_EX;
ac_reg<F_EX_WB> EX_WB;
ac_reg<F_WB_LW> WB_LW;

ac_stage FR,DE,EX,WB,LW;

```

Figura 4 – Descrição dos parâmetros do pipeline.

A divisão da execução entre os estágios deu origem ao novo `ppc405_ca-isa.cpp`, no qual cada comportamento conta com uma sequência de comandos específica de cada estágio. Conforme ilustra a figura 5, o comportamento específico da instrução `add` cumpre seu papel somente no estágio `EX`, pois os outros passos são dados em comportamentos superiores na hierarquia. Assim, o

instruction cuida da busca no FE, e da escrita no registrador de destino no WB. O XO1 cuida da leitura dos registradores no DE, adiantando-os de outros estágios, se necessário, para evitar perigo de dados.

```

//Instruction add behavior method.
void ac_behavior( add )
{
    switch( stage ) {
    case FE: {
        } break;

    case DE: {
        } break;

    case EX: {
        EX_WB.alures = ex_value1 + ex_value2;
        EX_WB.regwrite = DE_EX.regwrite;
        EX_WB.rdest = DE_EX.r3;

        if (DE_EX.oe==1)
            add_XRR_OV_SO_update(EX_WB.alures,ex_value1,ex_value2,0);
        if (DE_EX.rc==1)
            CRO_update(EX_WB.alures);

        } break;

    case WB: {
        } break;

    case LW: {
        } break;

    default: {
        } break;
    }
};

```

Figura 5 – Comportamento de uma instrução no modelo com precisão de ciclos.

4. RESULTADOS EXPERIMENTAIS

4.1. Configuração experimental

O computador utilizado nos experimentos possui um processador Pentium 4 com 3,0 GHz de frequência, cache L2 de 1 MB e memória principal de 1 GB. A execução foi feita no sistema operacional Debian GNU/Linux kernel 2.6. A versão do ArchC utilizado foi a 1.6.0 e a compilação das aplicações do benchmark para executarem no simulador se deu com o compilador redirecionável GCC 3.3.1 disponível no site do projeto ArchC [6]. Com a finalidade de gerar os binários para a versão 405 do PowerPC, passamos um parâmetro adicional ao compilador (-mcpu=405).

4.2. Validação dos modelos

Na fase de desenvolvimento inicial do modelo foi utilizado um suíte de programas chamado ac_stone, disponibilizado pela equipe do ArchC, com a finalidade de avaliar a coerência do modelo na execução de programas simples.

Foi feito um teste preliminar com o ac_stone onde foram corrigidos erros semânticos mais evidentes. Ele facilitou a identificação da instrução com erro pois

possibilitava a depuração do modelo com o gdb, uma funcionalidade especial do ArchC. Porém não foi possível fazer o mesmo com o modelo com precisão de ciclos, pois este não provê a possibilidade de uso do depurador.

Já na fase de validação com o MiBench utilizou-se um modelo de referência certificado do ArchC, o modelo do MIPS, considerado confiável para gerar os resultados esperados das aplicações do benchmark. Depois, cada aplicação foi executada no modelo do PowerPC405 para gerar um resultado, que é comparado com o do modelo de referência. Caso houvesse divergência, o modelo era revisado e testado novamente. A figura 6 mostra o diagrama da metodologia de validação utilizada. Este esquema foi usado com cada uma das aplicações do MiBench.

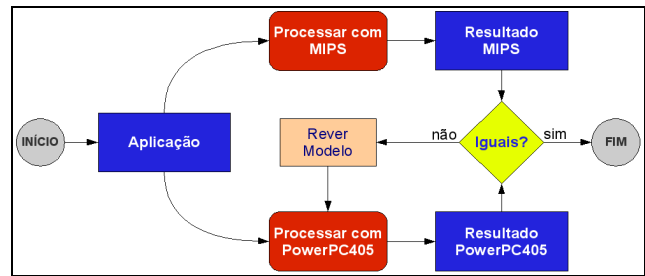


Figura 6 – Metodologia de Validação.

A Tabela 1 mostra os resultados obtidos com os programas do benchmark Mibench no modelo funcional. Os do modelo com precisão de ciclos estão na Tabela 2.

Tabela 1 – Resultados do modelo funcional

Nome do programa	Número de instruções	Instruções executadas	Tempo (s) simulação
bitcount	10.963	52.344.934	3,2
qsort	13.795	18.548.465	1,7
susan corners	18.767	3.412.539	0,4
susan edges	18.767	6.945.490	0,8
susan smoothing	18.767	27.946.125	1,9
jpeg encoder	29.735	25.513.629	1,9
jpeg decoder	34.577	6.541.842	0,5
dijkstra	13.630	50.840.786	3,4
patricia	14.387	316.690.011	32,1
rijndael coder	13.192	29.628.126	3,6
rijndael decoder	13.192	29.550.668	3,6
sha	10.299	12.041.205	0,7
adpcm encoder	9.728	30.873.714	2,3
adpcm decoder	9.726	26.276.530	2,1
crc32	10.397	28.930.492	1,5
gsm encode	21.005	20.154.916	1,6
gsm decode	21.005	11.960.574	0,8

É notável a grande divergência no tempo de execução de um modelo funcional para um com precisão de ciclos, agravado pelo fato da versão funcional ter sido executada de forma compilada, ou seja, a aplicação era compilada

Tabela 2 – Resultados do modelo com precisão de ciclos

Nome do programa	Número de instruções	Instruções executadas	Tempo (s) simulação
bitcount	10.963	66.774.644	5.347,6
qsort	13.795	25.735.539	2.274,7
susan corners	18.767	4.468.109	333,9
susan edges	18.767	9.086.674	778,8
susan smoothing	18.767	34.424.177	2.841,1
jpeg encoder	29.735	33.838.859	3.448,9
jpeg decoder	34.577	7.673.592	815,7
dijkstra	13.630	69.257.416	7.513,3
patricia	14.387	413.266.071	38.265,9
rijndael coder	13.192	32.665.360	2.676,8
rijndael decoder	13.192	32.627.232	2.660,3
sha	10.299	13.786.971	1.069,2
adpcm encoder	9.728	46.631.970	3.902,5
adpcm decoder	9.726	39.992.072	3.693,0
crc32	10.397	37.209.932	3.609,5
gsm encode	21.005	23.462.280	2.065,8
gsm decode	21.005	14.991.282	1.212,3

junto com o modelo, o que agiliza consideravelmente sua execução quando comparada à forma interpretada.

A quantidade de instruções buscadas do modelo com precisão de ciclos é cerca de 30% maior que no modelo funcional devido ao perigo de controle a que foi submetido o modelo temporizado, fazendo com que tenha que descartar instruções buscadas por causa de desvios de processamento.

5. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho extendemos o modelo puramente funcional do PowerPC, adicionando-lhe as instruções específicas da Unidade de Multiplicação-Soma do PowerPC405. Também descrevemos o modelo temporizado deste processador com a definição do seu *pipeline* e a divisão do fluxo de execução de cada instrução em estágios. Foram executadas aplicações típicas de sistemas embarcados nos modelos visando sua validação e medida de desempenho.

O modelo puramente funcional apresentou resultados satisfatórios. Porém o modelo com precisão de ciclos, apesar de executar corretamente as aplicações apresentou-se lento na simulação e necessitou de cerca de um terço a mais de busca de instruções devido a um fator já relatado na seção anterior.

6. AGRADECIMENTOS

Agradeço a Bruno Corsi dos Santos, Daniel Carlos Casarotto e José Otávio Carlomagno Filho, autores de modelos funcionais do PowerPC que foram utilizados como ponto de partida para os modelos aqui descritos.

REFERÊNCIAS

- [1] Moore G. E., *Cramming more components onto integrated circuits*, 1965.
- [2] Patterson, D., Hennessy, J., *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. Morgan Kaufmann Publishers, 2004.
- [3] PowerPC Processor Reference Guide - Embedded Development Kit, 2003.
Disponível em <http://www.xilinx.com>.
- [4] PPC405Fx Embedded Processor Core User's Manual, 2005. Disponível em <http://www.ibm.com>.
- [5] Sangiovanni-Vincentelli, A., Martin, G., *Platform-Based Design and Software Design Methodology for Embedded Systems*. IEEE Design & Test of Computers, NOV 2001.
- [6] The ArchC Architectural Description Language.
Disponível em <http://www.archc.org>.
- [7] The Open SystemC Initiative.
Disponível em <http://www.systemc.org>.