Universidade Federal de Mato Grosso do Sul
Faculdade de Computação
Curso de Ciência da Computação
Disciplina: Laboratório de Hardware
Professor: Ricardo R. Santos

Especificação do Trabalho 1 - 1ª Parte

1 Objetivo

Geral: implementar em VHDL a especificação a seguir do processador Pocket-MIPS Monociclo apresentada na Parte I e Parte II deste documento.

Específicos:

- Implementar em VHDL todos os elementos que compõem a via de dados do processador Pocket-MIPS Monociclo.
- Simular a implementação realizada a fim de detectar possíveis bugs e falhas.
- Construir TestsBenchs para o modelo do processador Pocket-MIPS Monociclo.
- Prototipar usando FPGA o processador implementado em VHDL.

2 Metodologia

- Entendimento de todo o funcionamento do processador Pocket-MIPS Monociclo
- Implementação das interfaces e arquiteturas de acordo com a especificação apresentada na Seção 3
- Documentação de todo o código implementado
- Projeto e construção de TestBenchs que testam o maior número possível de instruções e características implementadas
- Divisão do trabalho entre componentes do grupo
- Reuniões frequentes a fim de avaliar o andamento do trabalho

3 Processador Pocket-MIPS Monociclo

O Processador Pocket-MIPS Monociclo (PMM) é um processador totalmente baseado na máquina MIPS com implementação da via de dados monociclo. As instruções do PMM possuem apenas 1 byte, o ISA é composto por 10 instruções, há quatro registrados de propósito geral visíveis aos usuários (0..3)e cada registrador armazena apenas 1 byte. Há outros dois registradores (4..5) que não são visíveis aos usuários e também armazenam 1 byte.

A memória máxima endereçável é de 256 bytes e é acessada (endereçada) por byte.
A Figura 1 apresenta a via de dados do PMM.
Os circuitos presentes no *datapath* do PMM são:

- ULA

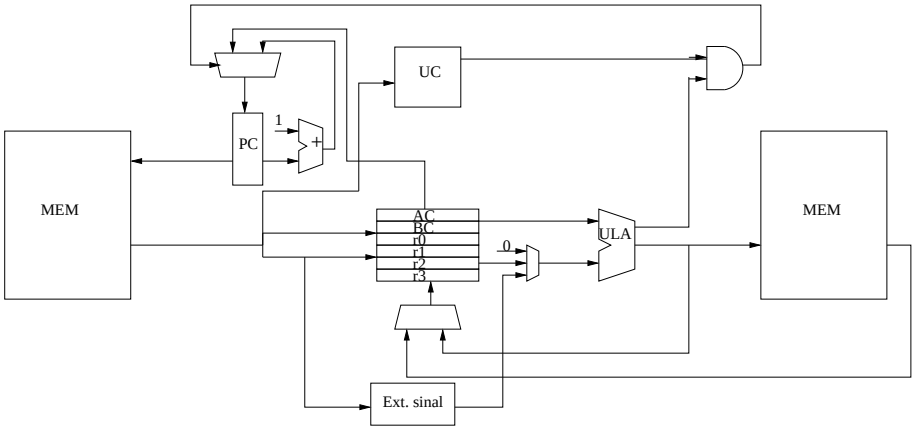

Figura 1: Via de Dados do Processador PMM

- Unidade de extensão de sinal
- Somadores adicionais
- Gerador de *clock*
- Registrador PC
- Conjunto de registradores
- Memória de instruções
- Memória de dados
- Unidade de controle principal

Circuitos auxiliares: multiplexadores, portas and e not.

As instruções presentes no PMM e seus formatos são apresentados na Tabela 1. Observe que o formato varia de acordo com o tipo da instrução (o símbolo 'X' indica *don't care*):

- Instruções tipo R: opcode, registrador origem, função
- Instruções tipo I: opcode, registrador origem, imediato OBS: o imediato é representado em complemento de 2.
- Instruções tipo J (inclui beq): opcode, registrador origem, registrador origem

3.1 Comportamento das Unidades do PMM

- ALU
  - Entidade: alu
  - Sinal de entrada: A, B: bit_vector(7 downto 0)
  - ALUOperation: bit_vector(1 downto 0)
  - Sinal de saída: Result: bit_vector(7 downto 0) e Zero: bit

Instrução	Exemplo	Significado	Formato
Instruções Tipo R			
ADD	ADD r1	Ac=Ac+r1	000 01 XXX
SUB	SUB r1	Ac=Ac-r1	010 01 XXX
MTA	MTA r1	Ac=r1+0	110 XXX 00
MFA	MFA r1	r1=Ac+0	111 XXX 00
MTB	MTB r1	Bc=r1+0	110 XXX 01
MFB	MFB r1	r1=Bc+0	111 XXX 01
HALT	HALT	finaliza programa	111 XXX 11
Instruções Tipo I			
ADDI	ADDI 6	Ac=Ac+6	001 00110
LW	LW r1,0	Ac=Mem(r1+0)	100 01 000
SW	SW r1,0	Mem(r1+0)=Ac	101 01 000
Instruções Tipo J			
BEQ	BEQ r0, r1	if (r0==r1) PC=Ac	011 00 01 X

Tabela 1: Conjunto de Instruções do PMM

  - Operação: sempre que algum sinal de entrada muda de valor, determina Result e Zero. Se A = B então Zero = '1'. Senão, Zero = '0'. Result = A op B, onde op é especificada por ALUOperation.

- Extensão de sinal
  - Entidade: sign_extend
  - Sinal de entrada: DataIn5: bit_vector(4 downto 0)
  - Sinal de saída: DataOut8: bit_vector(7 downto 0)
  - Operação: sempre que sinal de entrada muda de valor, determina DataOut8. DataOut8 possui valor de DataIn5 nos 5 bits menos significativos e bit 7 (bit de sinal) de DataIn5 nos 3 bits mais significativos.

- Adder
  - Entidade: adder
  - Sinais de entrada: A: bit_vector(7 downto 0), B: bit_vector(7 downto 0) (apenas para Adder2)
  - Sinal de saída: Sum: bit_vector(7 downto 0)
  - Operação e observações: sempre que algum sinal de entrada muda de valor, determina Sum como A + B. Adder não é ALU completa pois faz apenas soma.

- Clock
  - Entidade: clock_generator
  - Sinal de entrada: Halt : bit
  - Sinal de saída: Clk : bit
  - Operação: Gera continuamente o sinal de clock (ciclo do clock = 2 ns). Se Halt = '1', gera mais um ciclo do clock e suspende indefinidamente.

- Banco de registradores
  - Entidade: reg_file
  - Sinais de entrada: ReadRegister1: bit_vector(1 downto 0) (número do registrador a ser lido). ReadRegister2: bit_vector(1 downto 0) (número do registrador a ser lido). WriteRegister: bit_vector(1 downto 0) (número do registrador a ser escrito). WriteData: bit_vector(7 downto 0) (dado a ser escrito em registrador). RegWrite: bit (sinal de controle que habilita escrita em registrador). Clk: bit (sinal do clock)
  - Sinais de saída: ReadData1: bit_vector(7 downto 0) (dado lido de registrador). ReadData2: bit_vector(7 downto 0) (dado lido de registrador).
  - Operação: sempre que algum sinal de entrada muda de valor: lê registradores; se RegWrite = '1' e Clk está na borda de subida, escreve em registrador.
- PC
  - Entidade: pc_register
  - Sinais de entrada: Clk : bit, NewPC : bit_vector(7 downto 0)
  - Sinal de saída: CurrentPC : bit_vector(7 downto 0)
  - Operação: Inicialmente: CurrentPC = "00000000". Sempre que Clk está na borda de subida: CurrentPC=NewPC
- Unidade de Controle Principal
  - Entidade: control_unit
  - Sinal de entrada: Op : bit_vector(2 downto 0), Func(1 downto 0)
  - Sinais de saída: RegDst, RegWrite, MemRead, MemWrite, MemToReg, Branch, InvZero, Halt : bit. ALUSrc, ALUOp : bit_vector(1 downto 0)
  - Operação: Sempre que algum sinal de entrada muda de valor, determina sinais de saída.
- Memória de Instruções
  - Entidade: instruction_memory
  - Sinal de entrada: ReadAddress : bit_vector(7 downto 0)
  - Sinal de saída: Instruction: bit_vector(7 downto 0)
  - Operação: Possui estrutura de dados representando memória endereçada por bytes. Vetor de 256 bytes, indexados de 0 a 255. Cada byte é bit_vector(7 downto 0). **Inicialmente:** Lê arquivo com programa e carrega instruções na memória, a partir do endereço 0. Lê nome do arquivo do teclado. Cada instrução é seqüência de 8 bits. Sempre que sinal de entrada muda de valor, lê palavra da memória a partir do endereço ReadAddress, e coloca no sinal de saída Instruction.
- Memória de Dados
  - Entidade: data_memory
  - Sinais de entrada: Address : bit_vector(7 downto 0), WriteData : bit_vector(7 downto 0), MemRead : bit, MemWrite : bit, Clk : bit, Halt : bit

  - Sinal de saída: ReadData : bit_vector(7 downto 0)
  - Operação: Possui estrutura de dados representando memória endereçada por bytes: Vetor de 256 bytes, indexados de 0 a 255. Cada byte é bit_vector(7 downto 0).

- Sistema
  - Entidade: system
  - Sinal de entrada ou saída: nenhum
  - Operação: Possui instâncias de todos os demais módulos. Interliga todos os demais módulos, formando a via de dados. Possui circuitos auxiliares: multiplexadores, portas and e not.

4 Avaliação

- CHK: a avaliação da implementação constará das avaliações parcial do checkpoint (checkpoint terá peso 1);
- REL_SW: avaliação do relatório e implementação (peso 7);
- BONUS: bônus pela implementação em hardware, adição de novas instruções, adição de novos recursos visando ILP (peso 3).
- NOTA FINAL DO TRABALHO=(CHK*1+REL_SW*0,7+BONUS*0,3)

5 Cronograma

- 21/10: início do trabalho
- 16/11: aula para desenvolvimento do trabalho. Nessa aula, todos os alunos devem utilizar o horário para desenvolver atividades de implementação relacionadas ao trabalho.
- 18/11: checkpoint. Cada grupo deve apresentar para o professor o estágio atual de desenvolvimento do trabalho. Nessa data, espera-se que cada grupo apresente uma versão com implementação e teste de todas as instruções do tipo R.
- 09/12: entrega do trabalho.

6 Formato para Entrega do Trabalho

O trabalho deverá ser submetido eletronicamente através da página da disciplina no Moodle. O prazo de entrega se encerra no dia 09 de dezembro às 23h. O sistema de submissão será bloqueado nesse horário e não serão aceitas submissões por outros meios.

Cada grupo deverá entregar TODOS os arquivos fonte necessários para compilação e execução do código implementado compactados em um único arquivo com a extensão .zip ou .tar.gz. O arquivo compactado deverá possuir o seguinte formato de nome: T4_NOME_DOS_MEMBROS_GRUPO.tar.gz (exemplos: T4_Joao_Jose.tar.gz ou T4_Joao_Jose.zip). Junto aos arquivos fonte deverá acompanhar também um arquivo README informando:

1. nome do programa (executável), objetivo, autores, data da última modificação e versão;

2. pré-requisitos (outros comandos e suas respectivas versões) para instalação e utilização da ferramenta;

3. breve descrição dos diretórios que compõem a implementação;

4. comandos para compilação;

5. comandos para execução;

6. BUGs e falhas;

O grupo deve entregar um relatório de acordo com a formatação disponível em:

`http://www.sbc.org.br/index.php?language=1&subject=60&content=downloads.`

O relatório deve estar contido no arquivo compactado junto aos demais arquivos de código fonte (arquivos de entrada, arquivos .vhd e fontes em geral. OBS: não incluir arquivos de backup e ou arquivos objeto). O relatório deve possuir, no máximo, 10 páginas e poderá ser construído usando OpenOffice ou word ou Latex mas a entrega deverá ser feita usando o formato .pdf. O roteiro sugerido para o relatório é o seguinte:

1. Introdução: apresentar, brevemente, o objetivo do trabalho e o que foi implementado, apresentar a organização do relatório.

2. Arquitetura PMM e Implementação em VHDL: descrever, detalhadamente, os recursos que foram implementados (quantos e quais) do PMM. Informar se algum recurso não foi implementado totalmente. Informar se algum recurso extra (bônus) foi implementado.

3. Experimentos e Resultados: apresentar o funcionamento da implementação por meio de um exemplo.

4. Conclusões: reforçar o que foi implementado, indicar o que o trabalho prático contribuiu para o entendimento da disciplina como um todo ou de algum tópico específico da disciplina, apresentar as dificuldades encontradas na implementação do trabalho e quais são as próximas atividades que devem ser implementados visando finalizar a construção do compilador. Indicar sugestões para a melhoria da disciplina nos próximos anos.