UCP: Construindo um Caminho de Dados (Parte I)

Cristina Boeres

Instituto de Computação (UFF)

Fundamentos de Arquiteturas de Computadores

Material baseado cedido pela Profa. Fernanda Passos

Componentes Combinacionais *vs.* Sequenciais

A maioria dos componentes vistos são *combinacionais*

- A saída é uma função (combinação) das entradas
- Exemplos:
  - Somador
  - Deslocador
  - ALU

Já o resgistrador é um componente de outra classe

- Em um dado ciclo, a saída não depende das suas entradas
- O registrador mantém estado entre ciclos
- O valor de saída é o armazenado em um ciclo anterior
- A entrada só muda este estado (para os próximos ciclos)

Unidade de Controle *vs.* Caminho de Dados

- Podemos dividir o funcionamento de um processador em duas partes:
  - O Caminho de Dados
  - A Unidade de Controle
- O Caminho de Dados é o *hardware* que executa as instruções
  - Realiza processamento, armazenamento, ...
- A Unidade de Controle gerencia o caminho de dados
  - Diz a este o que fazer

O Caminho de Dados

- O Caminho de Dados contém componentes como:
  - Registradores
  - Multiplexadores
  - ALU
  - Somadores
- Tais componentes devem ser interligados de acordo com alguma lógica de processamento
- Objetivo:
  - Criar um ou mais caminhos para os dados que são transformados

A Unidade de Controle

- Vários componentes do Caminho de Dados possuem linhas de controle
  - um multiplexador escolhendo entre duas ou mais entradas
  - uma ALU efetuando uma operação específica

- **Alguém precisa determinar o estado destas linhas de controle**
  - Este é justamente o trabalho da **Unidade de Controle**.

A Unidade de Controle

Componentes típicos:

- Extensores de sinal
- Somadores
- **Decodificadores**
- etc

Seu trabalho é:

- Decodificar a instrução a ser executada
- Ativar (e desativar) as linhas de controle adequadas para esta execução

Projetando um Processador Simplificado

Funcionalidades de controle e do caminho de dados

- A partir dos componentes descritos anteriormente
- Como combiná-los e conectá-los de forma a implementar um processador funcional

Baseado em um subconjunto do conjunto de instruções do MIPS

- Conjunto pequeno, mas suficiente para executar programas complexos

Processo será *iterativo*

- Começaremos com versões simples
- Adicionaremos mais funcionalidades aos poucos

Etapas do Projeto

Passo 1: Definição do *conjunto de instruções*

- Deve ser pequeno
- Mas suficiente para rodarmos programas interessantes.

Passo 2: Definição de uma estrutura organizacional macroscópica

- Definição das características e quantidades de certos componentes
- Como o número de registradores
- E funcionalidades da ALU

Passo 3: Definição da interconexão destes componentes no caminho de dados

- Como conectá-los permitindo a execução das instruções desejadas

Definindo um Conjunto de Instruções

Para executar programas interessantes:

- Operações relevantes devem ser consideradas:
  - Ler/escrever dados de/para MP
  - Carregar constantes
  - Operações aritméticas
    - Soma, subtração, ...
  - Operações lógicas
    - Operações bit a bit, como *E* e *Ou*.
  - Comparações
  - Desvios para outros pontos do código
    - Condicionais e incondicionais

Simplificações

A primeira simplificação é no número de instruções

- O processador simplificado entenderá apenas 10 instruções

Apenas instruções sobre inteiros

- operações com ponto-flutuante pode ser implementado em *software*

Operações aritiméticas: apenas instruções de soma e subtração

- Multiplicação e divisão podem ser implementadas em *software*

Em termos de lógica, apenas duas operações: *E* e *Ou*

- Outras operações (como a negação, Xor,...) podem ser implementadas em *software*

Simplificações

Uma instrução para leitura e escrita de dados da MP:

- leitura: *load word* e escrita : *store word*

Uma instrução de comparação:

- *set less than*

Uma instrução de desvio condicional:

- *branch if equal*

uma instrução de desvio incondicional:

- *jump*
  - haverá uma perda de funcionalidade (chamada de procedimentos)

uma instrução para carregamento de constantes:

- *add immediate*
  - MIPS completo prevê várias instruções que executam operações lógicas/aritméticas sobre constantes

Resumo do Conjunto de Instruções (I)

Instrução	Operandos	Descrição
load word	Dois registradores e um imediato	Carrega palavra de memória da posição reg2 + imm para o registrador reg1
store word	Dois registradores e um imediato	Escreve valor da palavra em reg1 para a posição de memória reg2 + imm.
add	Três registradores	Soma os valores dos registradores reg2 e reg3 e armazena resultado em reg1.
subtract	Três registradores	Subtrai os valores dos registradores reg2 e reg3 e armazena resultado em reg1.
and	Três registradores	Realiza operação lógica AND bit a bit entre reg2 e reg3 e armazena resultado em reg1.
or	Três registradores	Realiza operação lógica AND bit a bit entre reg2 e reg3 e armazena resultado em reg1.

Resumo do Conjunto de Instruções (II)

Instrução	Operandos	Descrição
add immediate	Dois registradores e um imediato	Soma o valor do registrador reg2 ao valor do imediato e armazena resultado em reg1.
set on less than	Três registradores	Se o valor de reg2 é maior ou igual ao valor de reg3, escreve 0 em reg1. Escreve 1, caso contrário.
branch on equal	Dois registradores e um imediato	Se os valores de reg1 e reg2 forem iguais, salta para PC + 1 + imm (imediato em número de instruções).
jump	Um imediato	Salto incondicional para endereço dado pelo imediato multiplicado por 4.

Definindo o Formato das Instruções

- Uma outra etapa necessária no projeto de um processador é definir o formato das instruções.
  - Como cada instrução é codificada em bits.
  - Opcode, operandos, ...
- Vamos novamente simplificar nosso trabalho nesta etapa:
  - Usaremos os mesmos formatos de instrução do MIPS.

Formatos de Instrução do MIPS

Tipo	Formato (bits)					
R	opcode	rs	rt	rd	shamt	função
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I	opcode (6)	rs (5)	rt (5)	imediato (16)		
	6 bits	5 bits	5 bits	(16 bits)		
J	opcode (6)	endereço (26)				
	6 bits	26 bits				

- as instruções do MIPS usam um de três formatos
  - Formato depende do número e tipo dos operandos

- **Note que nosso subconjunto de instruções possui instruções de todos os três tipos**

Formatos Mais Específicos: Instruções Aritméticas/Lógicas

- Nosso subconjunto de instruções possui 6 instruções que realizam operações lógicas/aritméticas.
  - *add*, *subtract*, *set on less than*, *or*, *and* e *add immediate*.

- Destas, cinco operam sobre dois registradores e colocam o resultado em um terceiro registrador.
  - Exceção: instrução *add immediate*.
  - Um dos operandos é uma constante (imediato).

Instruções Aritméticas/Lógicas (II)

formato binário:

0000 00*ss ssst tttt dddd d*000 00*ff ffff*

- *sssss*: indica o registrador do primeiro operando
- *ttttt*: indica o registrador do segundo operando
- *ddddd*: indica o registrador do resultado
- *ffffff*: campo "função".

Instrução	Campo "Função"
add	100000
subtract	100010
and	100100
or	100101
set on less than	101010

Instruções Aritméticas/Lógicas (III)

- A única exceção é a instrução *add immediate*.
- Ela precisa de dois registradores e uma constante (imediato).
  - Logo, formato usado é o I.
- Mais especificamente:

$$0010\ 00ss\ ssst\ tttt\ iiii\ iiii\ iiii\ iiii$$

- Onde:
  - *sssss*: número inteiro positivo indicando registrador do primeiro operando.
  - *ttttt*: número inteiro positivo indicando registrador do resultado.
  - *iiiiiiiiiiiiiiii*: 16 bits indicando um número inteiro em Complemento a Dois.

Instruções Transferência de Memória

- Há duas:
  - *load word*, *store word*.
- Ambas têm os mesmos tipos de operandos.
  - Dois registradores e um imediato
  - Formato de instrução I
- Mais especificamente:

$$loadword \rightarrow 1000\ 11ss\ ssst\ tttt\ iiii\ iiii\ iiii\ iiii$$

$$storeword \rightarrow 1010\ 11ss\ ssst\ tttt\ iiii\ iiii\ iiii\ iiii$$

- Onde:
  - *sssss*: indica o registrador contendo o endereço base
  - *ttttt*: indica o registrador de/para onde o dado será lido/escrito
  - *iiiiiiiiiiiiiiii*: 16 bits indicando um número inteiro em Complemento a Dois.

Instruções de Desvio Condicional

- Há apenas uma: *branch on equal*.
- Recebe dois registradores e uma constante (imediato).
  - Formato I
- Mais especificamente:

0001 00*ss ssst tttt iiii iiii iiii iiii*

- Onde:
  - *sssss*: indica o registrador do primeiro operando
  - *ttttt*: indica o registrador do segundo operando
  - *iiiiiiiiiiiiiiii*: 16 bits indicando um número inteiro em Complemento a Dois

Instruções de Desvio Incondicional

- Como o desvio é incondicional, só precisamos de um operando:
  - Endereço para o qual saltar
  - Formato J
- Mais especificamente:

$$0000\ 10ii\ iiii\ iiii\ iiii\ iiii\ iiii\ iiii$$

- Onde:
  - *iiiiiiiiiiiiiiiiiiiiiiiiii*: 26 bits indicando um número inteiro **positivo**

Resumindo os Formatos

Instrução	Formato
add	0000 00*ss ssst tttt dddd* d000 0010 0000
subtract	0000 00*ss ssst tttt dddd* d000 0010 0010
and	0000 00*ss ssst tttt dddd* d000 0010 0100
or	0000 00*ss ssst tttt dddd* d000 0010 0101
set on less than	0000 00*ss ssst tttt dddd* d000 0010 1010
add immediate	0010 00*ss ssst tttt iiii iiii iiii iiii*
load word	1000 11*ss ssst tttt iiii iiii iiii iiii*
store word	1010 11*ss ssst tttt iiii iiii iiii iiii*
branch on equal	0001 00*ss ssst tttt iiii iiii iiii iiii*
jump	0000 10*ii iiii iiii iiii iiii iiii iiii*

Exercício

- Nesta aula falamos sobre o formato das instruções do nosso processador hipotético.
- Estamos usando o mesmo formato de instruções do MIPS.
- Particularmente, Todas as instruções têm um opcode de 6 bits.
- Responda:
  - Para o subconjunto de instruções que estamos implementando, seis bits são necessários? Justifique.
  - Em caso negativo, determine o número mínimo de bits para o opcode.