Universidade Federal do Rio Grande do Sul
Instituto de Informática

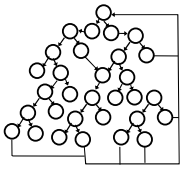
Organização de Computadores

Aula 08

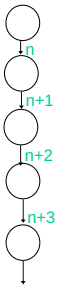
Bloco de controle multi-ciclo
Projeto com microprogramação

Alternativa a FSM para Multi-cicle?

- MIPS-lite tem 7 instruções, 10 estados na FSM
- Máquinas reais tem 100 ou mais instruções; o número de estados pode chegar a centenas ou mesmo milhares!!!
- Problema: FSM diagrama de bolotas muito grande



Mais observações



- Programas: próxima instrução é geralmente implícita
  - PC register determines instruction
  - next instruction always at PC+4 (unless branch or jump)
- FSM Controller: often only one exit arc from current state to next state
- Suppose borrow idea from Machine Language, represent each control step as some kind of "instruction"?
- Leads to Microprogrammed Control

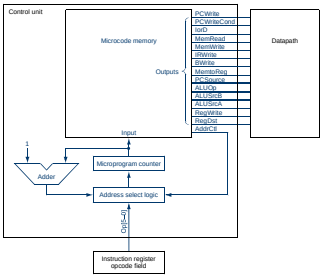
Micro-programmed Control

- In microprogrammed control, FSM states become microinstructions of a microprogram ("microcode")
  - one FSM state=one microinstruction
  - usually represent each micro-instruction textually, like an assembly instruction
- FSM current state register becomes the microprogram counter (micro-PC)
  - normal sequencing: add 1 to micro-PC to get next micro-instruction
  - microprogram branch: separate logic determines next microinstruction

Microprogramming Vs Hardwired Control

- **Microprogramming offers flexibility for design and architectural changes. The control memory (ROM) can be reprogrammed or replaced. Hardwired control is difficult to design for complex set architecture. Once it is designed, no further change is possible**
- **Microprogramming is slow because the control memory is accessed in every cycle. Memory access is slow. Hardwired control is fast because the cycle time depends on the combinational logic delay of the control unit, which is much less than memory access time.**

Microprogramming



- **What are the "microinstructions" ?**

Bloco de controle
projeto com microprogramação

1. Introdução

2. Formato das micro-instruções

3. Microprogramas

4. Implementação do bloco de controle

1. Introdução

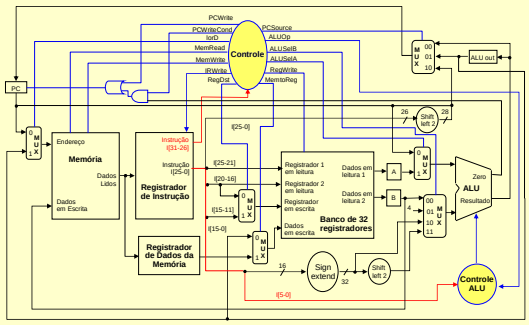
- **FSM é muito complexa para blocos de controle de processadores com conjuntos complexos de instruções**
  - **formatos variados de instruções**
  - **muitos modos de endereçamento**
  - **instruções com número variável de ciclos**
- **microprogramação: maneira estruturada de desenvolver um bloco de controle muito complexo**
  - **cada micro-instrução define os sinais de controle necessários para execução de um passo da instrução**
  - **microprograma é um conjunto de micro-instruções para a execução de uma ou mais instruções**
  - **microprograma é armazenado numa "memória de controle" (ROM ou PLA)**
- **microprograma pode ser desenvolvido simbolicamente**
  - **micro-assembler gera as micro-instruções em formato binário**

2. Formato das micro-instruções

- **formatos possíveis para micro-instruções**
  - **1 bit para cada sinal de controle – micro-instrução pode ficar muito larga**
  - **codificação por campos – cada campo deve especificar sinais de controle que nunca precisam ser gerados simultaneamente**
- **micro-instruções para o MIPS têm 7 campos**
  - ***ALU control* – operação na ALU**
  - ***SRC1* – fonte do 1º operando para ALU**
  - ***SRC2* – fonte do 2º operando para ALU**
  - ***Register control* – registrador a ser escrito com resultado da ALU**
  - ***Memory* – escrita ou leitura na memória e fonte de endereço;**

  registrador destino, em instrução *load*, ou registrador fonte, em instrução *store*
  - ***PCWrite control* – escrita no PC**
  - ***Sequencing* – seleção da próxima micro-instrução**

1. Sinais de controle



Formato das micro-instruções

ALU control	**Add**	**ALU executa uma soma**
	Func code	**ALU executa operação especificada no campo de função**
	Subt	**ALU executa subtração**
SRC1	**PC**	**PC é primeira entrada da ALU**
	A	**Registrador A é primeira entrada da ALU**
SRC2	**B**	**Registrador B é segunda entrada da ALU**
	4	**Constante 4 é segunda entrada para ALU**
	Extend	**Saída da unidade de extensão de sinal é 2ª entrada para ALU**
	Extshift	**Saída da unidade de deslocamento de 2 bits é 2ª entrada para ALU**
Register Control	**Read**	**Lê 2 registradores indicados em rs e rt e coloca em A e B**
	Write ALU	**Escreve conteúdo de ALUout no registrador indicado no campo rd**
	Write MDR	**Escreve conteúdo de MDR no registrador indicado no campo rt**
Memory	**Read PC**	**PC: endereço de leitura em memória; Resultado escrito em IR**
	Read ALU	**ALUout: endereço de leitura em memória; Resultado escrito em MDR**
	Write ALU	**ALUout: endereço de escrita em memória; Dado p/ escrita: registrador B**

Formato das micro-instruções

PCWrite control	**ALU**	**Escreve saída da ALU no PC**
	ALUout-cond	**Se saída Zero = 1, atualiza PC c/ conteúdo do registrador ALUout**
	Jump address	**Escreve endereço de desvio no PC**
Sequencing	**Seq**	**Escolhe a próxima microinstrução seqüencialmente**
	Fetch	**Volta para 1ª microinstrução para iniciar uma nova instrução**
	Dispatch i	**Dispatch usando a ROM especificada por "i" (1 ou 2)**

Tamanho da micro-instrução: 13 bits, supondo campos codificados

ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
2	1	2	2	2	2	2

Sinais de controle e os campos

- **Cada campo da micro-instrução gera um ou mais sinais de controle**
  - **ALU control : ALUOp**
  - **Src1 : AluSelA**
  - **Src2 : AluSelB**
  - **RegisterControl : RegDst, MemtoReg, RegWrite**
  - **Memory : IorD, MemRead, MemWrite, IRWrite**
  - **PC Write control : PCWrite, PCWriteCond, PCSource**
- **Para cada valor no campo, diferentes valores devem ser atribuídos aos sinais de controle**
  - **exemplo: campo Memory**

	IorD	MemRead	MemWrite	IRWrite
ReadPC	0	1	0	1
ReadALU	1	1	0	0
WriteALU	1	0	1	0

Valor Dispatch do campo Sequencing

- **seleciona endereço da próxima micro-instrução de acordo com entradas do bloco de controle**
- **tabela de endereços, usualmente armazenada em ROM, é indexada pelos sinais de entrada do BC**
- **de acordo com a FSM, esta situação ocorre nas transições a partir dos estados 1 e 2**

Tabela de Dispatch 1

op-code	próxima µinstrução
'lw' ou 'sw'	**2**
tipo R	**6**
'beq'	**8**
'jump'	**9**

Tabela de Dispatch 2

op-code	próxima µinstrução
'lw'	**3**
'sw'	**5**

3. Microprogramas

Busca da instrução, decodificação, cálculo de PC+4, cálculo de Target

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	**Add**	**PC**	**4**		**ReadPC**	**ALU**	**Seq**
	Add	**PC**	**Extshft**	**Read**			**Dispatch 1**

Primeira micro-instrução

ALU control, SRC1, SRC2	**calcular PC + 4**
Memory	**busca da instrução e escrita em IR**
PCWrite control	**saída da ALU é escrita em PC**
Sequencing	**seguir para próxima microinstrução**

Segunda micro-instrução

ALU control, SRC1, SRC2	**calcular PC + deslocamento x 4, estendido para 32 bits**
Register control	**usa rs e rt p/ ler os registradores; resultado em A e B**
Sequencing	**usar tabela 1 para obter endereço da próxima µinstrução**

Microprogramas

Instruções de referência à memória

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
LWSW1	**Add**	**A**	**Extend**				**Dispatch 2**
LW2					**ReadALU**		**Seq**
				WriteMDR			**Fetch**
SW2					**WriteALU**		**Fetch**

Instruções de tipo R

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Rform1	**Funct**	**A**	**B**				**Seq**
				WriteALU			**Fetch**

Microprogramas

Instrução de branch

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
BEQ1	Subt	A	B			ALUout-cond	Fetch

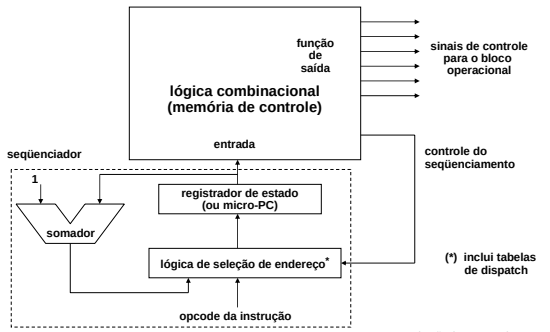
Instrução de jump

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Jump1						Jump address	Fetch

Memória de controle completa

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		ReadPC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
LWSW1	Add	A	Extend				Dispatch 2
LW2					ReadALU		Seq
				WriteMDR			Fetch
SW2					WriteALU		Fetch
Rform1	Funct	A	B				Seq
				WriteALU			Fetch
BEQ1	Subt	A	B			ALUout-cond	Fetch
Jump1						Jump address	Fetch

4. Implementação do BC



Microinstruction format

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
SRC2	B	ALUSrcB = 00	Register B is the second ALU input.
	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshift	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, IorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, IorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, IorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Maximally vs. Minimally Encoded

- **No encoding:**
  - **1 bit for each datapath operation**
  - **faster, requires more memory (logic)**
  - **used for Vax 780 — an astonishing 400K of memory!**
- **Lots of encoding:**
  - **send the microinstructions through logic to get control signals**
  - **uses less memory, slower**
- **Historical context of CISC:**
  - **Too much logic to put on a single chip with everything else**
  - **Use a ROM (or even RAM) to hold the microcode**
  - **It's easy to add new instructions**

Microcode: Trade-offs

- **Distinction between specification and implementation is sometimes blurred**
- **Specification Advantages:**
  - **Easy to design and write**
  - **Design architecture and microcode in parallel**
- **Implementation (off-chip ROM) Advantages**
  - **Easy to change since values are in memory (passado & futuro)**
  - **Can emulate other architectures passado & futuro!**
  - **Can make use of internal registers**
- **Implementation Disadvantages, SLOWER now that:**
  - **Control is implemented on same chip as processor**
  - **ROM is no longer faster than RAM**
  - **No need to go back and make changes (será?)**

Historical Perspective

- **In the '60s and '70s microprogramming was very important for implementing machines**
- **This led to more sophisticated ISAs and the VAX**
- **In the '80s RISC processors based on pipelining became popular**
- **Pipelining the microinstructions is also possible!**
- **Implementations of IA-32 architecture processors since 486 use:**
  - **"hardwired control" for simpler instructions (few cycles, FSM control implemented using PLA or random logic)**
  - **"microcoded control" for more complex instructions (large numbers of cycles, central control store)**
- **The IA-64 architecture uses a RISC-style ISA and can be implemented without a large central control store**

Pentium 4

- **Somewhere in all that "control we must handle complex instructions**
- **Processor executes simple microinstructions, 70 bits wide (hardwired)**
- **120 control lines for integer datapath (400 for floating point)**
- **If an instruction requires more than 4 microinstructions to implement, control from microcode ROM (8000 microinstructions)**
- **Its complicated!**

Sumário

- **If we understand the instructions…**
  We can build a simple processor!
- **If instructions take different amounts of time, multi-cycle is better**
- **Datapath implemented using:**
  - **Combinational logic for arithmetic**
  - **State holding elements to remember bits**
- **Control implemented using:**
  - **Combinational logic for single-cycle implementation**
  - **Finite state machine for multi-cycle implementation**

Próximos passos:

- **Já temos a máquina básica**
- **devemos aumentar o desempenho (o mercado assim o exige)**
- **devemos resolver os problemas que aparecerão no caminho**
  - **Desempenho do processador, memória, I/O, SO**