CSE 30321
Computer Architecture I

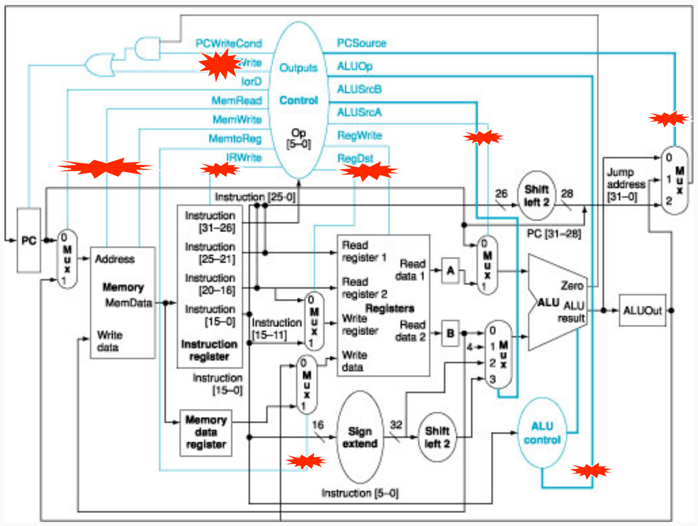
Lecture 16 - Multi Cycle Machines

Michael Niemier
Department of Computer Science and Engineering

Execution Sequence Summary

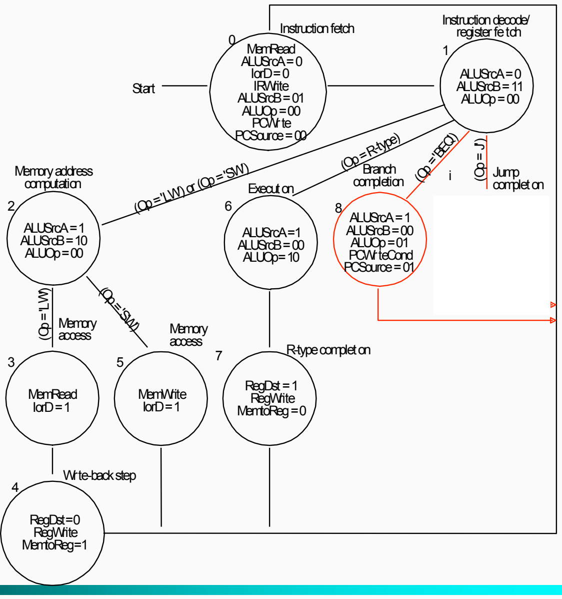
Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Mem[PC], PC = PC + 4			
Instruction decode/register fetch	A =RF [IR[25:21]], B = RF [IR[20:16]], ALUOut = PC + (sign-extend (IR[1:-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15:0])	if (A =B) then PC = ALUOut	PC = PC [31:28] | (IR[25:0]<<2)
Memory access or R-type completion	RF [IR[15:11]] = ALUOut	Load: MDR = Mem[ALUOut] or Store: Mem[ALUOut]= B		
Memory read completion		Load: RF[IR[20:16]] = MDR		

Control Signals



- **PC**: PCWrite, PCWriteCond, PCSource
- **Memory**: IorD, MemRead, MemWrite
- **Instruction Register**: IRWrite
- **Register File**: RegWrite, MemtoReg, RegDst
- **ALU**: ALUSrcA, ALUSrcB, ALUOp,

Finite State Diagram

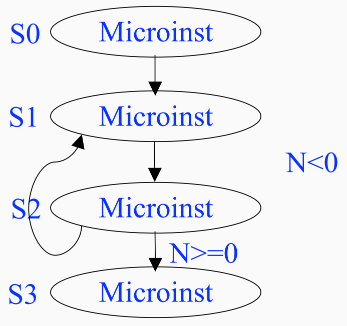

Start
Instruction fetch
0 MemRead ALUSrcA=0 IorD=0 IRWrite ALUSrcB=01 ALUOp=00 PCWrite PCSource=00
Instruction decode/ register fetch
1 ALUSrcA=0 ALUSrcB=11 ALUOp=00
(Op='LW) or (Op='SW)
(Op=R-type)
(Op='BEQ)
(Op='J')
Memory address computation
2 ALUSrcA=1 ALUSrcB=10 ALUOp=00
Execution
6 ALUSrcA=1 ALUSrcB=00 ALUOp=10
Branch completion
8 ALUSrcA=1 ALUSrcB=00 ALUOp=01 PCWriteCond PCSource=01
Jump completion
(Op='LW)
(Op='SW)
Memory access
3 MemRead IorD=1
Memory access
5 MemWrite IorD=1
R-type completion
7 RegDst=1 RegWrite MemtoReg=0
Write-back step
4 RegDst=0 RegWrite MemtoReg=1

Microprogramming as an Alternative

- Control unit can easily reach thousands of states with hundreds of different sequences.
  - A large set of instructions and/or instruction classes (x86)
  - Different implementations with different cycles per instruction
- Flexibility may be needed in the early design phase
- How about borrowing the ideas from what we just learned?
  - Treat the set of control signals to be asserted in a state as an *instruction* to be executed (referred to as microinstructions)
  - Treat state transitions as an instruction sequence
  - Define formats (mnemonics)
  - Specify control signals symbolically using microinstructions

Microprogramming as an Alternative (cont'd)

- Each state => one microinstruction
- State transitions => microinstruction sequencing
- Setting up control signals => executing microinstructions
- To specify control, we just need to write microprograms (or microcode)



Microinstruction Format (1)

- Group the control signals according to how they are used
- For the 5-cycle MIPS organization:
  - Memory: IorD, MemRead, MemWrite
  - Instruction Register: IRWrite
  - PC: PCWrite, PCWriteCond, PCSource
  - Register File: RegWrite, MemtoReg, RegDst
  - ALU: ALUSrcA, ALUSrcB, ALUOp
- Group them as follows:
  - Memory (for both Memory and Instruction Register)
  - PC write control (for PC)
  - Register control (for Register File)
  - ALU control } (for ALU)
  - SRC1 }
  - SRC2 }
  - Sequencing

Microinstruction Format (2)

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Sub	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's func 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	ALUSrc = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB= 10	Use output of the sign ext unit as the 2nd ALU input.
	Extshft	ALUSrcB= 11	Use output of shift-by-two unit as the 2nd ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR 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.

Microinstruction Format (3)

Field name	Value	Signals active	Comment
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 a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Sample Microinstruction (1)

- IFetch: IR = Mem[PC], PC = PC+4

PCWrite:	1
PCWriteCond:	
IorD:	0
MemRead:	1
IRWrite:	1
MemtoReg:	
PCSource:	
ALUOp:	00
ALUSrcB:	01
ALUSrcA:	0
RegWrite:	
RegDst:	
AddrCtrl:	11

Microinstruction:

ALUctrl	SRC1	SRC2	RegCtrl	Memory	PCWrite	Sequen
Add	PC	4	--	ReadPC	ALU	Seq

Sample Microinstruction (2)

- Decode: A= RF[IR[25:21]], B= RF[IR[20:16]],
  ALUOut = PC + Sign_Ext(IR[15:0]) << 2);

PCWrite:	
PCWriteCond:	
IorD:	
MemRead:	
IRWrite:	
MemtoReg:	
PCSource:	
ALUOp:	00
ALUSrcB:	11
ALUSrcA:	0
RegWrite:	
RegDst:	
AddrCtrl:	01

Microinstruction:

ALUctrl	SRC1	SRC2	RegCtrl	Memory	PCWrite	Sequen
Add	PC	ExtShf	Read			Disp 1

We'll talk about soon.

Sample Microinstruction (3)

- BEQ1: -> Ifetch,
  if (A=B) then PC= ALUout

PCWrite:	
PCWriteCond:	1
IorD:	
MemRead:	
IRWrite:	
MemtoReg:	
PCSource:	01
ALUOp:	01
ALUSrcB:	00
ALUSrcA:	1
RegWrite:	
RegDst:	
AddrCtrl:	00

Microinstruction:

ALUctrl	SRC1	SRC2	RegCtrl	Memory	PCWrite	Sequen
Sub	A	B			ALUout-cond	Fetch

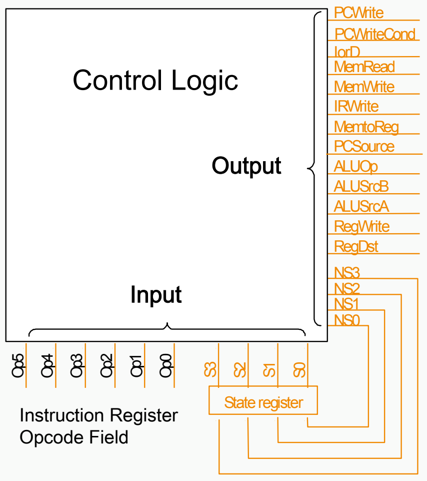
Put It All Together

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Sub	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

- What would a microassembler do?

Control Implementations

- The big picture:



- How to implement the control logic?
  – Random logic, memory-baed, mux-based, ...

Memory-Based Implementation

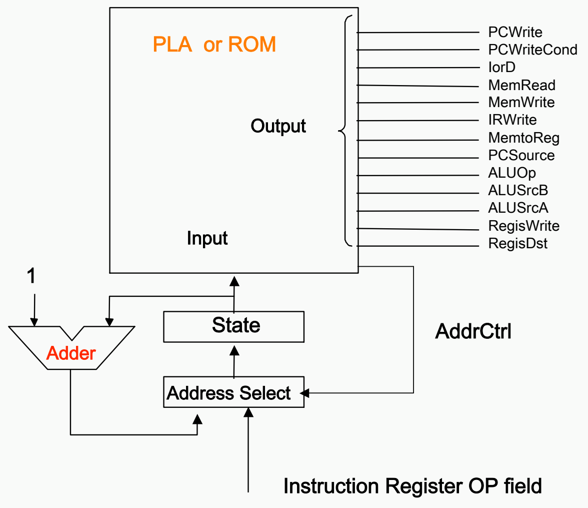
- ❑ Important factors to consider when using a memory:
  - How many address lines?
  - How many output bits?
  - So the ROM size is
- ❑ How many entries (or addresses) contain distinct values?
  - many outputs are the same or don’t cares so can be rather wasteful

Alternatives for Control Implementation

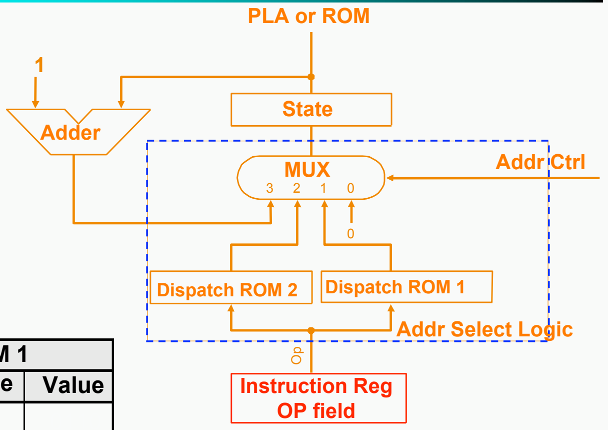
- ❑ Use hardwired random logic
  - Efficient especially if you have a good CAD tool (not Xilins, ok)
  - Not as flexible as memory-based
- ❑ Can we do better?
  - Use an explicit sequencer so avoid storing unused entries

Explicit Sequencer

- How many input lines?
  -
- How many output lines?
  - No. of control outs + No. of AddrCtrl



Address Select Logic



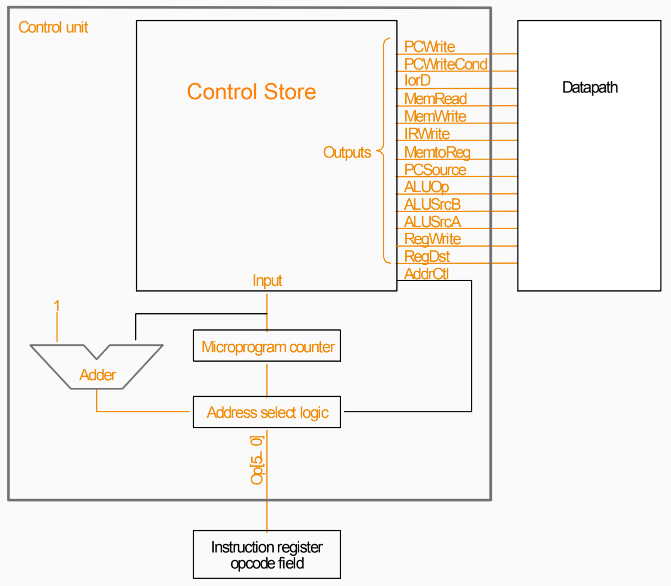
Dispatch ROM 1		
Op	Opcode name	Value
000000	R-format	
000010	jmp	
000100	beq	
100011	lw	
101011	sw	

Dispatch ROM 2		
Op	Opcode name	Value
100011	lw	
101011	sw	

Address Control Action

State No.	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

The Complete Design



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
  - Can emulate other architectures
  - 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

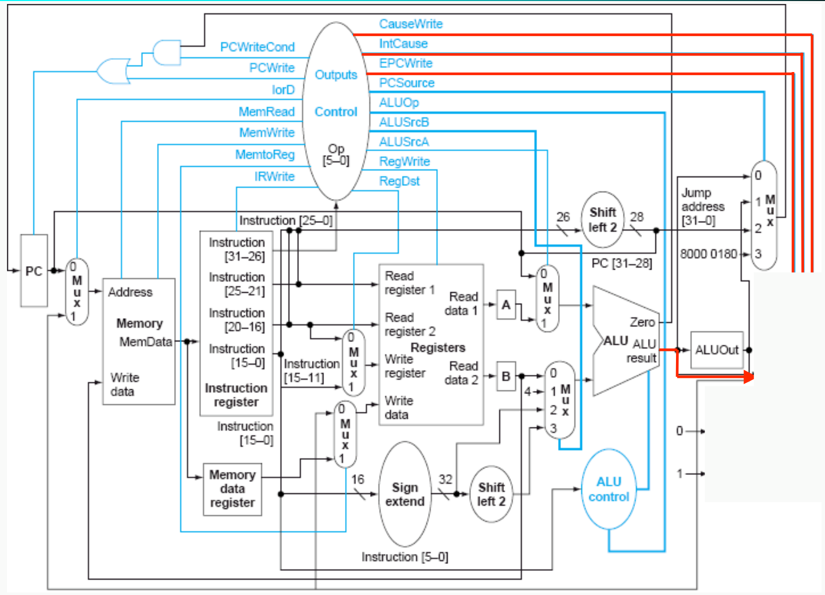
Exceptions

- Exceptions: unexpected events from <u>within</u> the processor
  - arithmetic overflow
  - undefined instruction
  - switching from user program to OS
- Interrupts: unexpected events from <u>outside</u> of the processor
  - I/O request
- Consequence: alter the normal flow of instruction execution
- Key issues:
  - detection
  - action
    - save the address of the offending instruction in the EPC
    - transfer control to OS at some specified address
- Exception type indication:
  - status register
  - interrupt vector

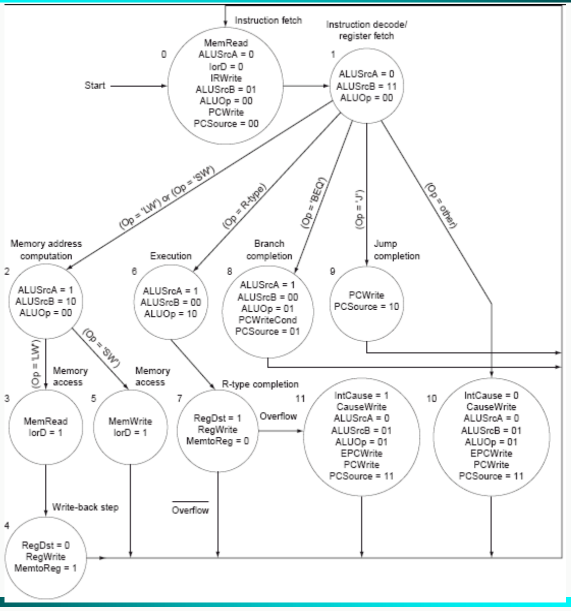
Exception Handling

- Types of exceptions considered:
  - undefined instruction
  - arithmetic overflow
- MIPS implementation:
  - EPC: 32-bit register, EPCWrite
  - Cause register: 32-bit register, CauseWrite
    - undefined instruction: Cause register = 0
    - arithmetic overflow: Cause register = 1
  - IntCause: 1 bit control
  - Exception Address: C0000000 (hex)
- Detection:
  - undefined instruction: op value with no next state
  - arithmetic overflow: overflow from ALU
- Action:
  - set EPC and Cause register
  - set PC to Exception Address

Datapath with Exception Handling



FSM with Exception Handling


Instruction fetch
Instruction decode/ register fetch
Start
0
MemRead ALUSrcA = 0 IorD = 0 IRWrite ALUSrcB = 01 ALUOp = 00 PCWrite PCSource = 00
1
ALUSrcA = 0 ALUSrcB = 11 ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address computation
2
ALUSrcA = 1 ALUSrcB = 10 ALUOp = 00
Execution
6
ALUSrcA = 1 ALUSrcB = 00 ALUOp = 10
Branch completion
8
ALUSrcA = 1 ALUSrcB = 00 ALUOp = 01 PCWriteCond PCSource = 01
Jump completion
9
PCWrite PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory access
Memory access
R-type completion
3
MemRead IorD = 1
5
MemWrite IorD = 1
7
RegDst = 1 RegWrite MemtoReg = 0
Overflow
11
IntCause = 1 CauseWrite ALUSrcA = 0 ALUSrcB = 01 ALUOp = 01 EPCWrite PCWrite PCSource = 11
10
IntCause = 0 CauseWrite ALUSrcA = 0 ALUSrcB = 01 ALUOp = 01 EPCWrite PCWrite PCSource = 11
Write-back step
Overflow
4
RegDst = 0 RegWrite MemtoReg = 1