CSE 490/590 Computer Architecture

MIPS Review & Pipelining I

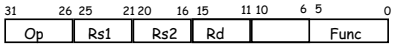
Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time…

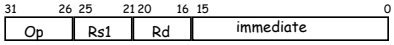
- An ISA can have multiple implementations
- (Briefly) CISC vs. RISC
  - CISC: microcoded (macro instructions & microinstructions)
  - RISC: combinational logic
- MIPS microarchitecture
  - Good RISC example
  - Fixed format instructions
  - Load/store architecture with single address mode
  - Simple branch
- Will continue on this today…

MIPS Instruction Formats

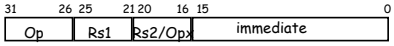
Register-Register



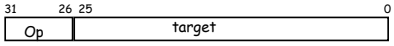
Register-Immediate



Branch (Same format as Reg-Imm)

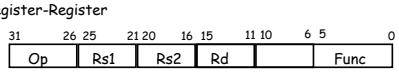


Jump / Call



Reg-Reg Instructions

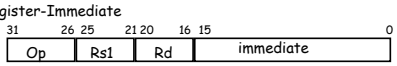
Register-Register



- Op (e.g., 0) encodes that this is a reg-reg instruction
- Func encodes the datapath operations (add, sub, etc.)
- Rd ← (Rs1) Func (Rs2)
- ADD R1,R2,R3
  - Add
  - Reg[R1] ← Regs[R2] + Regs[R3]

Reg-Imm Instructions

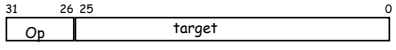
Register-Immediate



- Rd ← (Rs1) Op (Imm)
- ADDI R1,R2,#3
  - Add immediate
  - Regs[R1] ← Regs[R2] + 3
- LD R1,30(R2)
  - Load word
  - Regs[R1] ← Mem[30 + Regs[R2]]
- BEQZ R4, name
  - Branch equal zero
  - If (Regs[R4] == 0) PC ← name

Jump / Call

Jump / Call

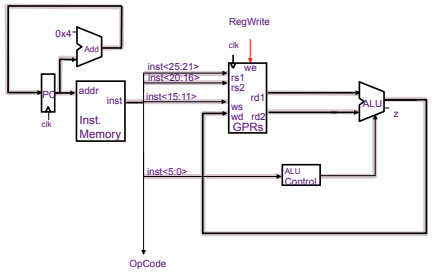


- Rd ← (Rs1) Op (Imm)
- J name
  - Jump
  - $PC_{6...31}$ ← name
- JAL name
  - Jump and link
  - Regs[R31] ← PC + 4; $PC_{6...31}$ ← name

Implementing MIPS:

Single-cycle per instruction datapath & control logic

Datapath: Reg-Reg ALU Instructions



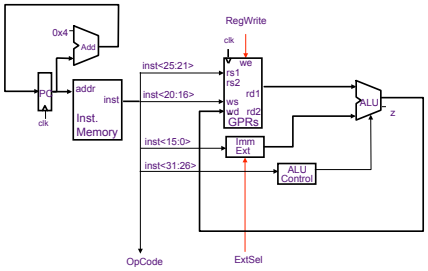
RegWrite Timing?

6	5	5	5	5	6
0	rs	rt	rd	0	func

31 26 25 21 20 16 15 11 5 0

rd ← (rs) func (rt)

Datapath: Reg-Imm ALU Instructions

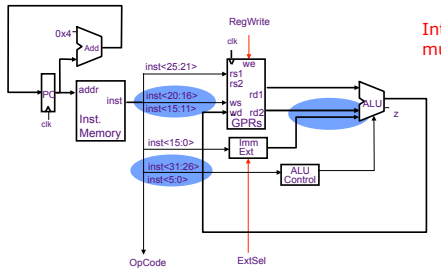


6	5	5	16
opcode	rs	rt	immediate

31 26 25 21 20 16 15 0

rt ← (rs) op immediate

Conflicts in Merging Datapath



Introduce muxes

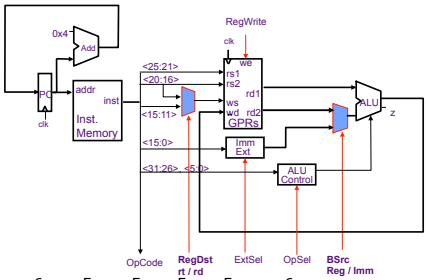
6	5	5	5	5	6
0	rs	rt	rd	0	func

rd ← (rs) func (rt)

opcode	rs	rt	immediate

rt ← (rs) op immediate

Datapath for ALU Instructions



6	5	5	5	5	6
0	rs	rt	rd	0	func

rd ← (rs) func (rt)

opcode	rs	rt	immediate

rt ← (rs) op immediate

Datapath for Memory Instructions

Should program and data memory be separate?

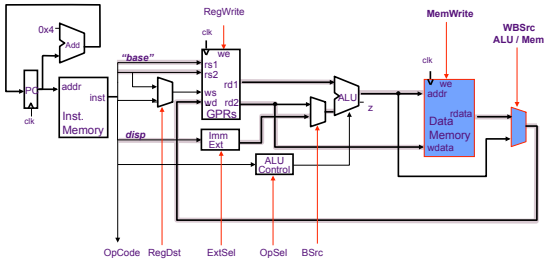
Harvard style: separate (Aiken and Mark 1 influence)
- read-only program memory
- read/write data memory

- Note:
  Somehow there must be a way to load the program memory

Princeton style: the same (von Neumann's influence)
- single read/write memory for program and data

- Note:
  A Load or Store instruction requires accessing the memory more than once during its execution

Load/Store Instructions: *Harvard Datapath*



6	5	5	16
opcode	rs	rt	displacement

31 26 25 21 20 16 15 0

addressing mode
(rs) + displacement

rs is the base register
rt is the destination of a Load or the source for a Store

CSE 490/590 Administrivia

- Please purchase a BASYS2 board (100K) as soon as possible.
  - Projects should be done individually.
- Quiz 1
  - Fri, 2/4
  - Closed book, in-class
  - 10%
- Class cancelled on Fri, 4/15
  - Will update the schedule

MIPS Control Instructions

Conditional (on GPR) PC-relative branch

6	5	5	16	
opcode	rs		offset	BEQZ, BNEZ

Unconditional register-indirect jumps

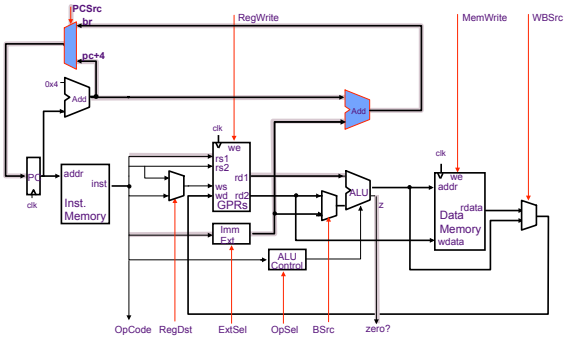
6	5	5	16	
opcode	rs			JR, JALR

Unconditional absolute jumps

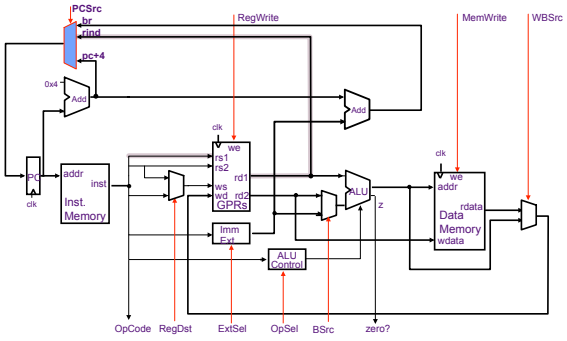
6	26	
opcode	target	J, JAL

- PC-relative branches add offset×4 to PC+4 to calculate the target address (offset is in words): ±128 KB range
- Absolute jumps append target×4 to PC<31:28> to calculate the target address: 256 MB range
- jump-&-link stores PC+4 into the link register (R31)

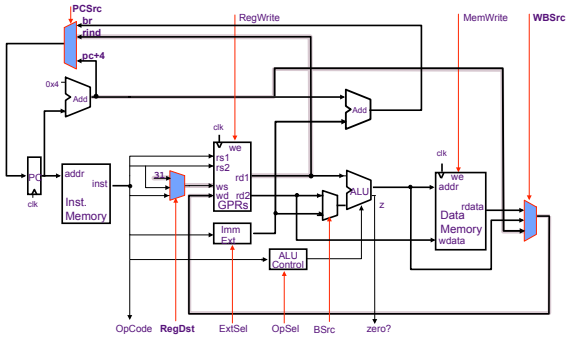
Conditional Branches (BEQZ, BNEZ)



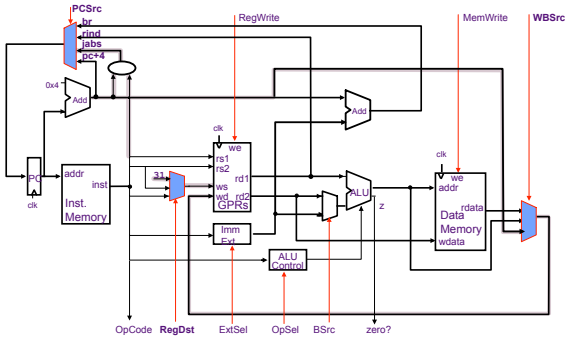
Register-Indirect Jumps (JR)



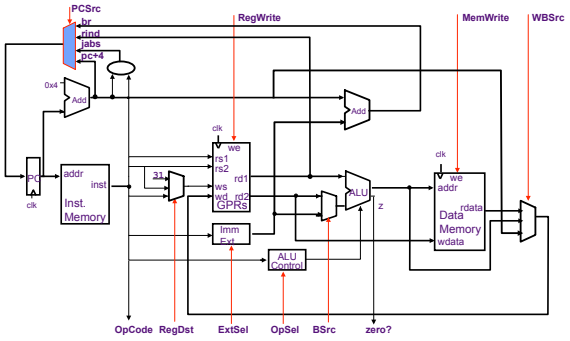
Register-Indirect Jump-&-Link (JALR)



Absolute Jumps (J, JAL)



Harvard-Style Datapath for MIPS



Single-Cycle Hardwired Control:

Harvard architecture

We will assume

- clock period is sufficiently long for all of the following steps to be "completed":

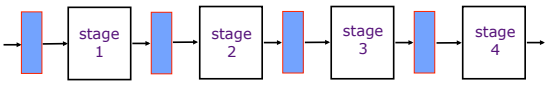
  1. instruction fetch
  2. decode and register fetch
  3. ALU operation
  4. data fetch if required
  5. register write-back setup time

  $\Rightarrow \quad t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB}$

- At the rising edge of the following clock, the PC, the register file and the memory are updated

Pipelined MIPS

An Ideal Pipeline



- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal
- The scheduling of an object entering the pipeline is not affected by the objects in other stages

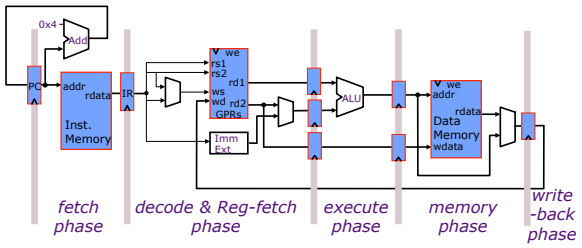
These conditions generally hold for industrial assembly lines.
But can an instruction pipeline satisfy the last condition?

Pipelined MIPS

To pipeline MIPS:

- First build MIPS without pipelining with CPI=1
- Next, add pipeline registers to reduce cycle time while maintaining CPI=1

Pipelined Datapath



Clock period can be reduced by dividing the execution of an instruction into multiple cycles

$t_C > \max \{t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW}\}$ (= t_{DM} *probably*)

However, CPI will increase unless instructions are pipelined

"Iron Law" of Processor Performance

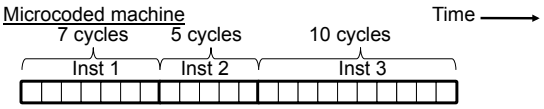
$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- Instructions per program depends on source code, compiler technology, and ISA
- Cycles per instructions (CPI) depends upon the ISA and the microarchitecture
- Time per cycle depends upon the microarchitecture and the base technology

Microarchitecture	CPI	cycle time
Microcoded	>1	short
Single-cycle unpipelined	1	long
Pipelined	1	short

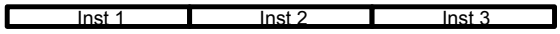
CPI Examples

Microcoded machine



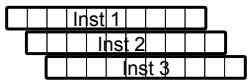
3 instructions, 22 cycles, CPI=7.33

Unpipelined machine



3 instructions, 3 cycles, CPI=1

Pipelined machine



3 instructions, 3 cycles, CPI=1

Technology Assumptions

- A small amount of very fast memory (caches) backed up by a large, slower memory
- Fast ALU (at least for integers)
- Multiported Register files (slower!)

Thus, the following timing assumption is reasonable

$t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW}$

A 5-stage pipeline will be the focus of our detailed design

- some commercial designs have over 30 pipeline stages to do an integer add!

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252