CS 61C: Great Ideas in Computer Architecture (Machine Structures)
Single-Cycle Processor Design

Instructors:
Randy H. Katz
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/fa10

Agenda

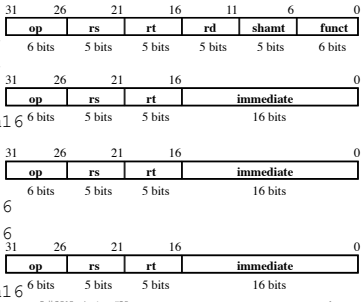
- MIPS-lite Datapath
- Administrivia
- Technology Break
- CPU Timing

Agenda

- MIPS-lite Datapath
- Administrivia
- Technology Break
- CPU Timing

The MIPS-lite Subset

- ADDU and SUBU
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`
- OR Immediate:
  - `ori rt,rs,imm16`
- LOAD and STORE Word
  - `lw rt,rs,imm16`
  - `sw rt,rs,imm16`
- BRANCH:
  - `beq rs,rt,imm16`



Register Transfer Language (RTL)

- RTL gives the meaning of the instructions

```
{op , rs , rt , rd , shamt , funct} ← MEM[ PC ]
{op , rs , rt ,   Imm16} ← MEM[ PC ]
```

- All start by fetching the instruction

```
Inst   Register Transfers
ADDU   R[rd] ← R[rs] + R[rt]; PC ← PC + 4
SUBU   R[rd] ← R[rs] – R[rt]; PC ← PC + 4
ORI    R[rt] ← R[rs] | zero_ext(Imm16); PC ← PC + 4
LOAD   R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4
STORE  MEM[ R[rs] + sign_ext(Imm16) ] ← R[rt]; PC ← PC + 4
BEQ    if ( R[rs] == R[rt] )
           then PC ← PC + 4 + (sign_ext(Imm16) || 00)
           else PC ← PC + 4
```

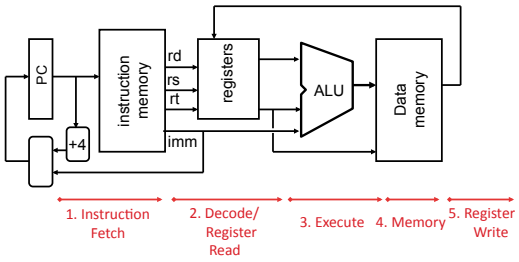

Processor Design Process

- Five steps to design a processor:
  - Step 1: Analyze instruction set to determine datapath requirements
  - Step 2: Select set of datapath components & establish clock methodology
  - Step 3: Assemble datapath components that meet the requirements
  - Step 4: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer
  - Step 5: Assemble the control logic

Step 1: Requirements of the Instruction Set

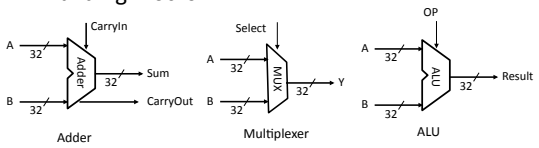
- Memory (MEM)
  - Instructions & data (will use one for each)
- Registers (R: 32 x 32)
  - Read RS
  - Read RT
  - Write RT or RD
- PC
- Extender (sign/zero extend)
- Add/Sub/OR unit for operation on register(s) or extended immediate
- Add 4 (+ maybe extended immediate) to PC
- Compare registers?

Generic Steps of Datapath



Step 2: Components of the Datapath

- Combinational Elements
- Storage Elements + Clocking Methodology
- Building Blocks



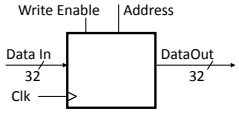
ALU Needs for MIPS-lite + Rest of MIPS

- Addition, subtraction, logical OR, ==:

```
ADDU  R[rd] = R[rs] + R[rt]; ...
SUBU  R[rd] = R[rs] - R[rt]; ...
ORI   R[rt] = R[rs] | zero_ext(Imm16)...
BEQ   if ( R[rs] == R[rt] )...
```

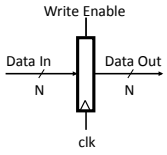
- Test to see if output == 0 for any ALU operation gives == test. How?
- P&H also adds AND, Set Less Than (1 if A < B, 0 otherwise)
- ALU follows Chapter 5

Storage Element: Idealized Memory



- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is found by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block: Address valid ⇒ Data Out valid after "access time"

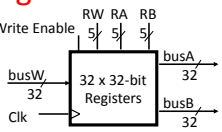
Storage Element: Register (Building Block)



- Similar to D Flip Flop except
  - N-bit input and output
  - Write Enable input
- Write Enable:
  - Negated (or deasserted) (0): Data Out will not change
  - Asserted (1): Data Out will become Data In on positive edge of clock

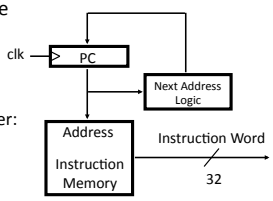
Storage Element: Register File

- Register File consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (clk)
  - Clk input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid ⇒ busA or busB valid after "access time."



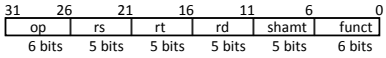
Step 3: Assemble DataPath Meeting Requirements

- Register Transfer Requirements ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
- Common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← "something else"

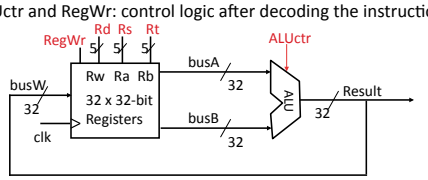


Step 3: Add & Subtract

- `R[rd] = R[rs] op R[rt] (addu rd,rs,rt)`
  - Ra, Rb, and Rw come from instruction's Rs, Rt, and Rd fields



  - ALUctr and RegWr: control logic after decoding the instruction



- … Already defined the register file & ALU

Agenda

- MIPS-lite
- Administrivia
- Technology Break
- CPU Design

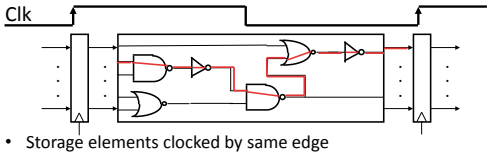
Agenda

- MIPS-lite
- Administrivia
- Technology Break
- CPU Design

Agenda

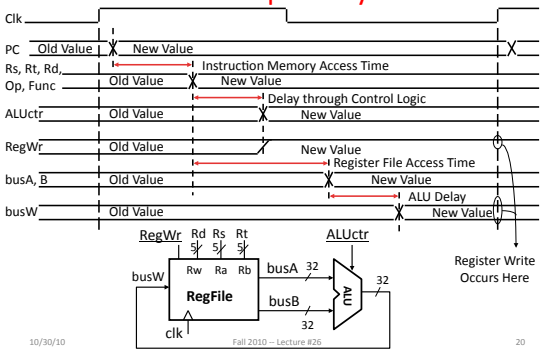
- MIPS-lite
- Administrivia
- Technology Break
- CPU Timing

Clocking Methodology



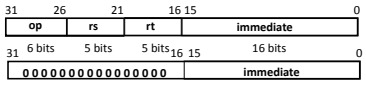
- Storage elements clocked by same edge
- Flip-flops (FFs) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay
- "Critical path" (longest path through logic) determines length of clock period

Register-Register Timing: One Complete Cycle

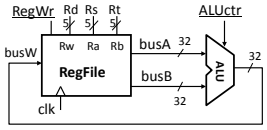


Logical Operations with Immediate

- R[rt] = R[rs] op ZeroExt[imm16]

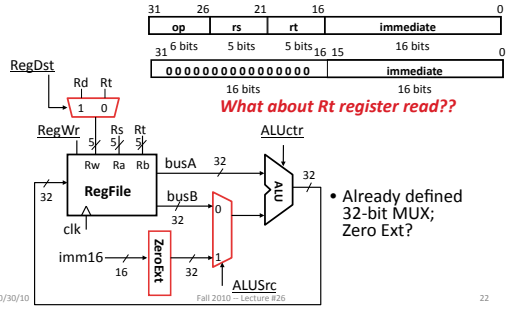


But we're writing to Rt register??



Logical Operations with Immediate

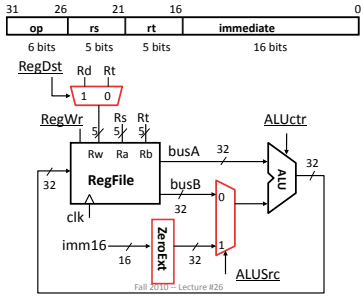
- R[rt] = R[rs] op ZeroExt[imm16]



- Already defined 32-bit MUX; Zero Ext?

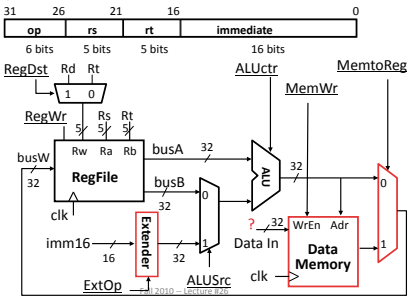
Load Operations

- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`

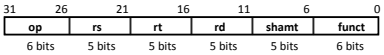


Load Operations

- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



RTL: The Add Instruction

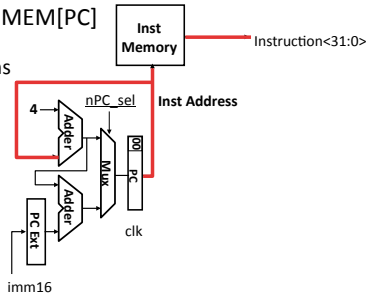


`add rd, rs, rt`

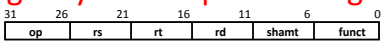
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at the Beginning of Add

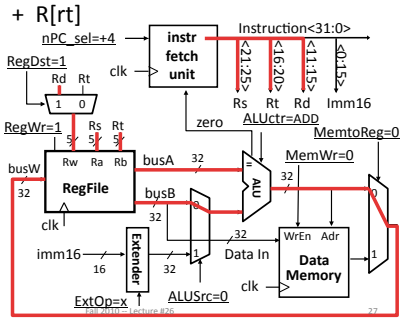
- Fetch the instruction from Instruction memory:
  Instruction = MEM[PC]
  - same for all instructions



Single Cycle Datapath during Add

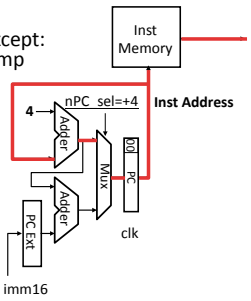


R[rd] = R[rs] + R[rt]

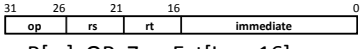


Instruction Fetch Unit at End of Add

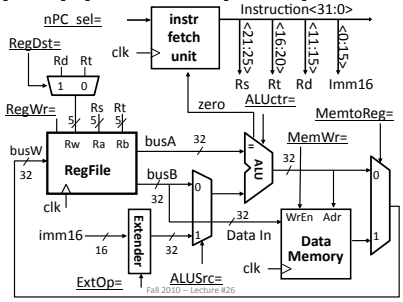
- PC = PC + 4
  - Same for all instructions except: Branch and Jump



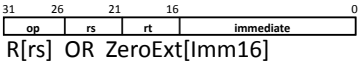
Single Cycle Datapath during Or Immediate



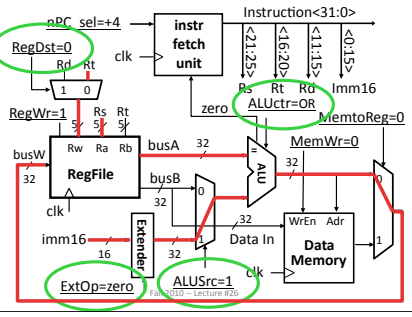
- R[rt] = R[rs] OR ZeroExt[Imm16]



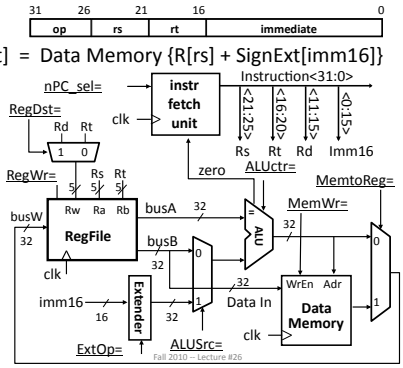
Single Cycle Datapath during Or Immediate



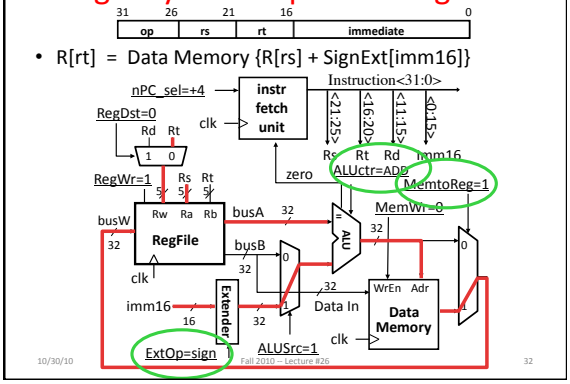
- R[rt] = R[rs] OR ZeroExt[Imm16]



Single Cycle Datapath during Load



Single Cycle Datapath during Load



Summary: Datapath's Control Signals

- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "ADD", "SUB", "OR"
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register