CS 61C: Great Ideas in Computer Architecture (Machine Structures)
Datapath Design

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

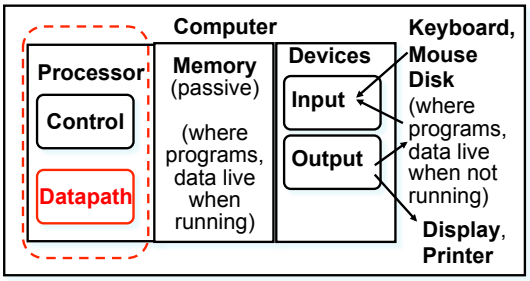
Agenda

- Stages of the Datapath
- Administrivia
- Technology Break
- Datapath Design

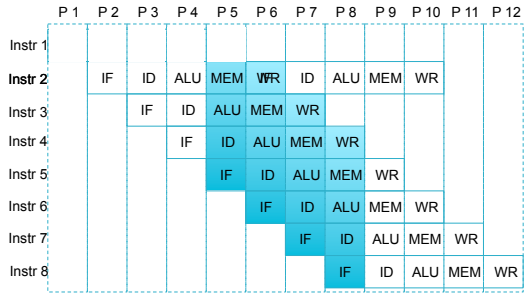
Agenda

- Stages of the Datapath
- Administrivia
- Technology Break
- Datapath Design

Five Components of a Computer



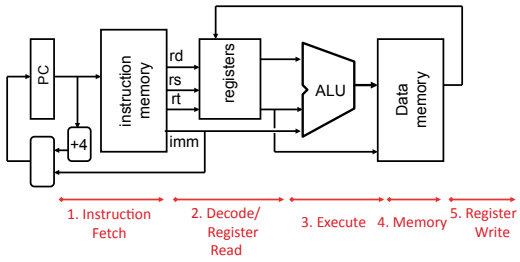
Instruction Level Parallelism



Five Stages of the Datapath

- Stage 1: *Instruction Fetch*
- Stage 2: *Instruction Decode*
- Stage 3: *ALU* (Arithmetic-Logic Unit)
- Stage 4: *Memory Access*
- Stage 5: *Register Write*

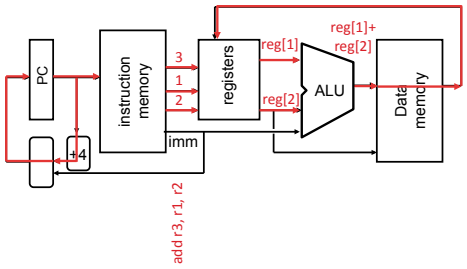
Generic Steps of Datapath



Datapath Walkthroughs (1/3)

- `add $r3,$r1,$r2 # r3 = r1+r2`
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is an `add`, then read registers `$r1` and `$r2`
  - Stage 3: add the two values retrieved in Stage 2
  - Stage 4: idle (nothing to write to memory)
  - Stage 5: write result of Stage 3 into register `$r3`

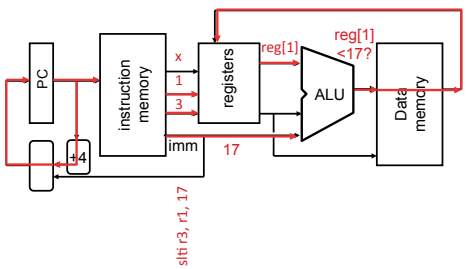
Example: `add` Instruction



Datapath Walkthroughs (2/3)

- `slti $r3,$r1,17`
  `# if (r1 <17 )r3 = 1 else r3 = 0`
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is an `slti`, then read register `$r1`
  - Stage 3: compare value retrieved in Stage 2 with the integer 17
  - Stage 4: idle
  - Stage 5: write the result of Stage 3 (1 if reg source was less than signed immediate, 0 otherwise) into register `$r3`

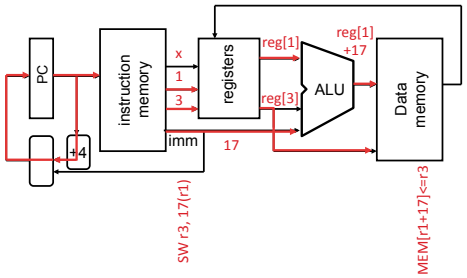
Example: `slti` Instruction



Datapath Walkthroughs (3/3)

- `sw $r3,17($r1) # Mem[r1+17]=r3`
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is a `sw`, then read registers `$r1` and `$r3`
  - Stage 3: add `17` to value in register `$r1` (retrieved in Stage 2) to compute address
  - Stage 4: write value in register `$r3` (retrieved in Stage 2) into memory address computed in Stage 3
  - Stage 5: idle (nothing to write into a register)

Example: `sw` Instruction



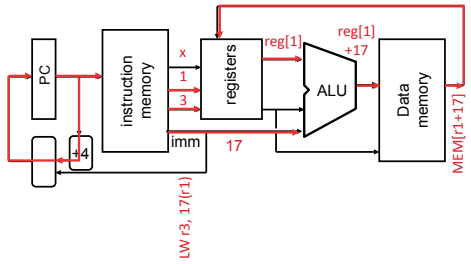
Why Five Stages? (1/2)

- Could we have a different number of stages?
  - Yes, and other architectures do
- So why does MIPS have five if instructions tend to idle for at least one stage?
  - Five stages are the union of all the operations needed by all the instructions.
  - One instruction uses all five stages: the load

Why Five Stages? (2/2)

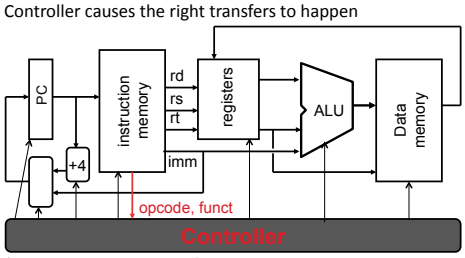
- `lw $r3,17($r1) # r3=Mem[r1+17]`
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is a `lw`, then read register `$r1`
  - Stage 3: add `17` to value in register `$r1` (retrieved in Stage 2)
  - Stage 4: read value from memory address computed in Stage 3
  - Stage 5: write value read in Stage 4 into register `$r3`

Example: `lw` Instruction



Datapath Summary

- Datapath based on data transfers required to perform instructions
- Controller causes the right transfers to happen



Agenda

- Stages of the Datapath
- Administrivia
- Technology Break
- Datapath Design

Administrivia

- Reordering of Projects #3 and #4
  - Project #3: Performance Improvement
    - Specification is up
    - Due 11/13, extra credit through 11/30
  - Project #4: Logic Design of a CPU
    - Due 11/27 (Saturday of Thanksgivings Weekend)
- Check out the calendar on the course web site

Agenda

- Stages of the Datapath
- Administrivia
- Technology Break
- Datapath Design

Agenda

- Stages of the Datapath
- Administrivia
- Technology Break
- Datapath Design

What Hardware Is Needed? (1/2)

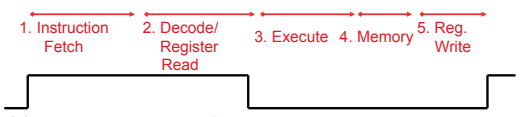
- PC: a register that keeps track of address of the *next* instruction to be fetched
- General Purpose Registers
  - Used in Stages 2 (Read) and 5 (Write)
  - MIPS has 32 of these
- Memory
  - Used in Stages 1 (Fetch) and 4 (R/W)
  - Caches makes these stages as fast as the others (on average, otherwise multicycle stall)

What Hardware Is Needed? (2/2)

- ALU
  - Used in Stage 3
  - Performs all necessary functions: arithmetic, logicals, etc.
- Miscellaneous Registers
  - One stage per clock cycle: Registers inserted between stages to hold intermediate data and control signals as they travels from stage to stage
  - Note: Register is a general purpose term meaning something that stores bits. Realize that not all registers are in the "register file"

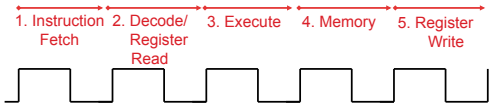
CPU Clocking (1/2)

- For each instruction, how do we control the flow of information though the datapath?
- Single Cycle CPU: All stages of an instruction completed within one long clock cycle
  - Clock cycle sufficiently long to allow each instruction to complete all stages without interruption within one cycle



CPU Clocking (2/2)

- Alternative multiple-cycle CPU: only one stage of instruction per clock cycle
  - Clock is made as long as the slowest stage



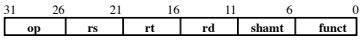
  - Several significant advantages over single cycle execution: Unused stages in a particular instruction can be skipped OR instructions can be pipelined (overlapped)

Processor Design

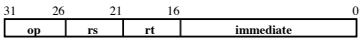
- Analyze instruction set architecture (ISA) to determine datapath requirements
  - Meaning of each instruction is given by register transfers
  - Datapath must include storage element for ISA registers
  - Datapath must support each register transfer
- Select set of datapath components and establish clocking methodology
- Assemble datapath components to meet requirements
- Analyze each instruction to determine sequence of control point settings to implement the register transfer
- Assemble the control logic to perform this sequencing

The MIPS-lite Subset

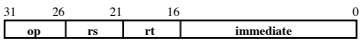
- ADDU and SUBU
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`



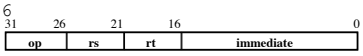
- OR Immediate:
  - `ori rt,rs,imm16`



- 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
```


Summary

- CPU design involves Datapath, Control
  - 5 Stages for MIPS Instructions
    1. Instruction Fetch
    2. Instruction Decode & Register Read
    3. ALU (Execute)
    4. Memory
    5. Register Write
- Datapath timing: single long clock cycle or one short clock cycle per stage
- Register transfer representation