Processor Architecture

From memory

Datapath

To memory

Control Signals

Status Signals

PI

PO
Where Does It All Begin?

• In a register called *program counter (PC).*
• PC contains the memory address of the next instruction to be executed.
• In the beginning, PC contains the address of the memory location where the program begins.
Where is the Program?

Processor

Program counter (register)

Memory

Start address

Machine code of program
How Does It Run?

Start
PC has memory address where program begins

Fetch instruction word from memory address in PC and increment PC → PC + 4 for next instruction

Decode and execute instruction

Program complete?

No

Yes → STOP
Datapath and Control

- Datapath: Memory, registers, adders, ALU, and communication buses. Each step (fetch, decode, execute) requires communication (data transfer) paths between memory, registers and ALU.

- Control: Datapath for each step is set up by control signals that set up dataflow directions on communication buses and select ALU and memory functions. Control signals are generated by a control unit consisting of one or more finite-state machines.
Datapath for Instruction Fetch

- **PC** → Address → **Instruction Memory**
- **Add** with 4
- Instruction word to control unit and registers
Register File: A Datapath Component

- **Read registers**
  - 5 -> reg 1
  - 5 -> reg 2

- **Write register**
  - 5 -> reg 1

- **Write data**
  - 32 -> reg 1 data
  - 32 -> reg 2 data

- **RegWrite from control**
  - 32

- **32 Registers (reg. file)**
Multi-Operation ALU

Operation select | ALU function
---|---
000 | AND
001 | OR
010 | Add
110 | Subtract
111 | Set on less than

Operation select from control

zero = 1, when all bits of result are 0
R-Type Instructions

- Also known as arithmetic-logical instructions
- add, sub, slt
- Example: add $t0, $s1, $s2
  - Machine instruction word
    
    \begin{align*}
    000000 & 10001 10010 01000 00000 100000 \\
    \text{opcode} & $s1 \quad $s2 \quad $t0 \quad \text{function}
    \end{align*}
  - Read two registers
  - Write one register
  - Opcode and function code go to control unit that generates RegWrite and ALU operation code.
Datapath for R-Type Instruction

000000 10001 10010 01000 00000 100000
opcode $s1 $s2 $t0 function (add)

Read register numbers
Write reg. number
Write data

32 Registers (reg. file)

RegWrite from control activated

ALU

Operation select from control (add)

zero

overflow

result

000000 10001 10010 01000 00000 100000
opcode $s1 $s2 $t0 function (add)
Load and Store Instructions

- I-type instructions
- `lw $t0, 1200 ($t1)` # incr. in bytes
  \[
  \begin{array}{ccccccccc}
  100011 & 01001 & 01000 & 0000 & 0100 & 1011 & 0000 \\
  \text{opcode} & $t1 & $t0 & 1200
  \end{array}
  \]

- `sw $t0, 1200 ($t1)` # incr. in bytes
  \[
  \begin{array}{ccccccccc}
  101011 & 01001 & 01000 & 0000 & 0100 & 1011 & 0000 \\
  \text{opcode} & $t1 & $t0 & 1200
  \end{array}
  \]
Datapath for lw Instruction

- **Opcode:** 100011
- **Register:** $t1, $t0
- **Address:** 010000 1011 0000
- **MemWrite:**
  - Read data
  - Write data
  - MemRead activated
- **MemRead:**
  - Addr.
  - Data memory
- **ALU:**
  - Operation select from control (add)
  - Zero
  - Result
  - Overflow
- **RegWrite from control activated:**
- **Data memory:**
- **Write data:**
- **Write reg. number:**
- **Write reg. number:**
- **Read register numbers:**
- **32 Registers (reg. file):**
  - $t1
  - $t0
  - 32
- **Sign extend:**
  - 16
- **RegWrite:**
  - 32
  - MemWrite
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead:**
  - 32
  - MemRead
- **MemWrite:**
  - 32
  - MemWrite
- **MemRead**
Datapath for sw Instruction

101011 01001 01000 0000 0100 1011 0000
opcode $t1 $t0 1200

Read register numbers
Write reg. number
Write data

32 Registers (reg. file)

$0100 0100 1011 0000

01000 01001 0100 0000
5 5 5

MemWrite activated
MemRead

Addr.
Data memory

ALU

zero
result

overflow

32
RegWrite from control

16

Data memory

Write data

01000 01001 0100

$0100 0100 1011 0000

1200

MemRead

08 Apr 2013
Computer Architecture@IIT Mandi
14
Branch Instruction (I-Type)

- `beq $s1, $s2, 25` # if $s1 = $s2, advance PC through 25 instructions

Note: Can branch within ± $2^{15}$ words from the current instruction address in PC.
Datapath for beq Instruction

Operation select from control (subtract)

32 Registers (reg. file)

Read register numbers

Write reg. number

Write data

RegWrite from control

Sign extend

Shift left 2

ALU

000100 10001 10010 0000 0000 0001 1001

opcode $s1 $s2

To branch control logic

PC+4

Branch target

From instruction fetch datapath

zero

result

overflow

32

32

32

32

32

Sign extend

08 Apr 2013 Computer Architecture@IIT Mandi
J-Type Instruction

- \text{j 2500} \# \text{jump to instruction 2,500}

\begin{center}
\begin{tabular}{l}
\text{opcode} \\
\text{2,500} \\
\text{32-bit jump address} \\
\text{0000 0000 0000 0000 0010 0111 0001 0000}
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{l}
\text{bits 28-31 from PC+4} \\
\text{000010 0000 0000 0000 0010 0111 0001 00}
\end{tabular}
\end{center}

26-bits
Datapath for Jump Instruction

```
PC  | 32  | Address | 32  | Instruction Memory
    | 32  |        |     |
Add | 32  | 32     | PC+4|
Branch addr. | 32  | 4      | 32  | 0 mux |
Branch |      |        | 0   |
0       | 1    | 32     | 32  | Jump |
Jump | 32  | 4      | 32  | 0 mux |
0       | 1    | 32     | 28  | 4  |
26     | 32  | 6      | 32  |
Instruction word to control and registers
Shift left 2
opcode (bits 26-31) to control
```
Combined Datapaths

- PC
- Instr. mem.
- Add
- Control
- ALU
- Data mem.
- Branch
- MemtoReg
- MemRead
- MemWrite
- RegDst
- Zero
- 0-15
- 0-5
- 0-25
- Shift left 2
- Jump
- 11-15
- 16-20
- 21-25
- 26-31
- Sign ext.
- Shift left 2
- ALU Cont.
Control Logic

Instruction
bits 26-31
opcode

Control Logic

RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite

Instruction
bits 0-5
funct.

ALU Control
to ALU
Control Logic: Truth Table

<table>
<thead>
<tr>
<th>Instr type</th>
<th>Inputs: instr. opcode</th>
<th>Outputs: control signals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31 30 29 28 27 26</td>
<td>RegDst Jump ALUSrc MemtoReg RegWrite MemRead MemWrite Branch ALOOp1 ALUOp2</td>
</tr>
<tr>
<td>R</td>
<td>0 0 0 0 0 0</td>
<td>1 0 0 0 1 0 0 0 0 1 0</td>
</tr>
<tr>
<td>lw</td>
<td>1 0 0 0 1 1</td>
<td>0 0 1 1 1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>sw</td>
<td>1 0 1 0 1 1</td>
<td>X 0 1 X 0 0 1 0 0 0</td>
</tr>
<tr>
<td>beq</td>
<td>0 0 0 1 0 0</td>
<td>X 0 0 X 0 0 0 1 0 1</td>
</tr>
<tr>
<td>j</td>
<td>0 0 0 0 1 0</td>
<td>X 1 X X X X X X X X X X X</td>
</tr>
</tbody>
</table>
How Long Does It Take?

- Assume control logic is fast and does not affect the critical timing. Major time delay components are ALU, memory read/write, and register read/write.

- Arithmetic-type (R-type)
  - Fetch (memory read): 2 ns
  - Register read: 1 ns
  - ALU operation: 2 ns
  - Register write: 1 ns
  - Total: 6 ns
Time for lw and sw (I-Types)

- ALU (R-type) 6ns
- Load word (I-type)
  - Fetch (memory read) 2ns
  - Register read 1ns
  - ALU operation 2ns
  - Get data (mem. Read) 2ns
  - Register write 1ns
  - Total 8ns
- Store word (no register write) 7ns
## Time for `beq` (I-Type)

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU (R-type)</td>
<td>6</td>
</tr>
<tr>
<td>Load word (I-type)</td>
<td>8</td>
</tr>
<tr>
<td>Store word (I-type)</td>
<td>7</td>
</tr>
<tr>
<td>Branch on equal (I-type)</td>
<td></td>
</tr>
<tr>
<td>- Fetch (memory read)</td>
<td>2</td>
</tr>
<tr>
<td>- Register read</td>
<td>1</td>
</tr>
<tr>
<td>- ALU operation</td>
<td>2</td>
</tr>
<tr>
<td>- Total</td>
<td>5</td>
</tr>
</tbody>
</table>
## Time for Jump (J-Type)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU (R-type)</td>
<td>6</td>
</tr>
<tr>
<td>Load word (I-type)</td>
<td>8</td>
</tr>
<tr>
<td>Store word (I-type)</td>
<td>7</td>
</tr>
<tr>
<td>Branch on equal (I-type)</td>
<td>5</td>
</tr>
<tr>
<td>Jump (J-type)</td>
<td></td>
</tr>
<tr>
<td>Fetch (memory read)</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>2</td>
</tr>
</tbody>
</table>
How Fast Can the Clock Be?

• If every instruction is executed in one clock cycle, then:
  – Clock period must be at least 8ns to perform the longest instruction, i.e., \( lw \).
  – This is a single cycle machine.
  – It is slower because many instructions take less than 8ns but are still allowed that much time.

• Method of speeding up: Use multicycle datapath.
Thank You