UMBC
AN HONORS UNIVERSITY

CMSC 411 | Lectures 13

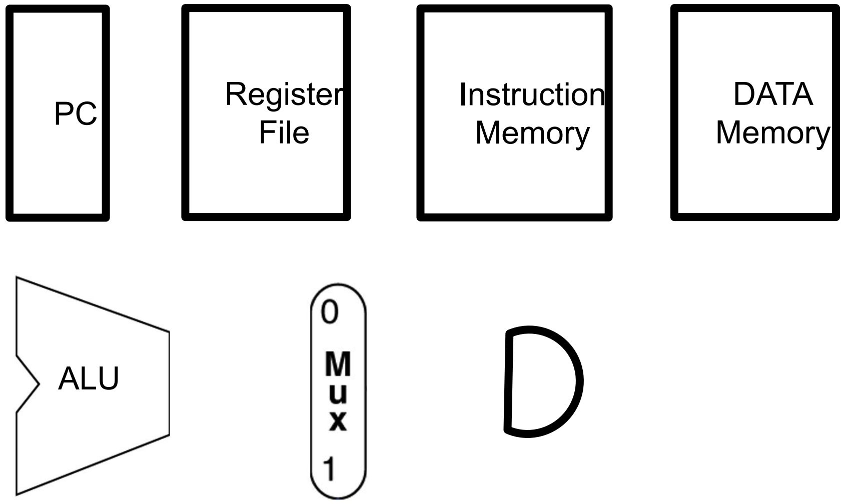
Single Cycle Datapath and Control Part I: Fundamentals

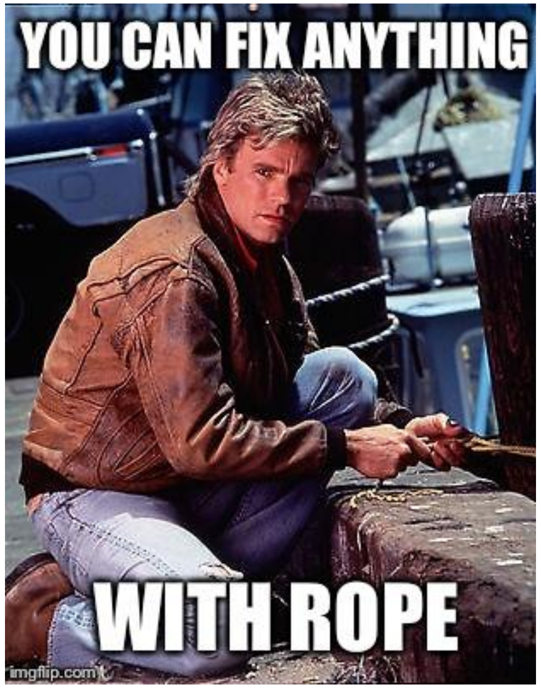
Ergun Simsek

Topics

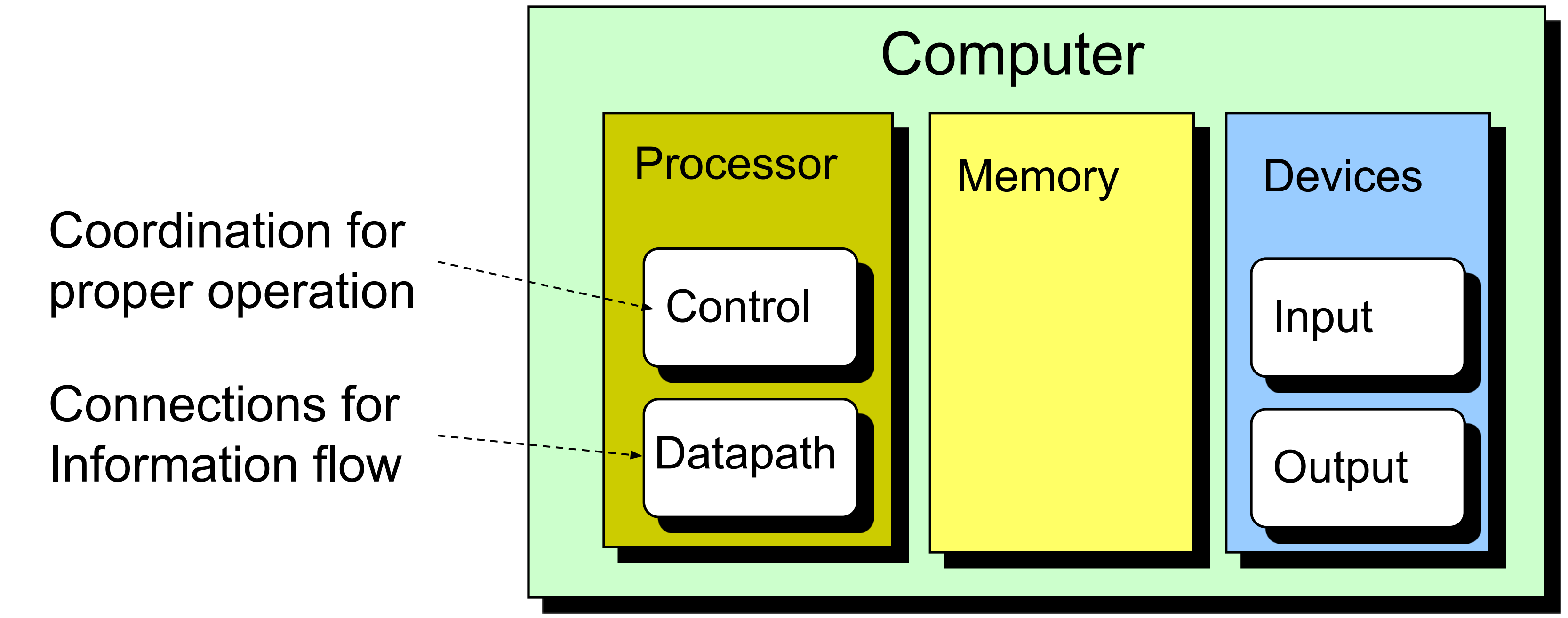
What does a microprocessor do?

How to build a simple microprocessor?

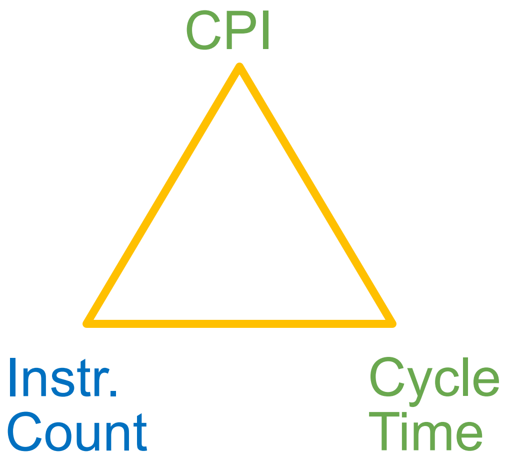




Introduction


Computer
Processor
Memory
Devices
Coordination for proper operation
Control
Input
Connections for Information flow
Datapath
Output

- So far, we discussed the performance and instruction set architecture
- Performance of a machine is determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction (CPI)
- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction



How to Design a Processor: step-by-step

0. Understand what a (or “the”) processor does

1. Analyze instruction set => datapath requirements
   - the meaning of each instruction is given by the *register transfers*
   - datapath must include storage element for ISA registers possibly more
   - datapath must support each register transfer

2. Select a set of datapath components and establish clocking methodology

3. Assemble datapath that meets the requirements

4. Analyze the implementation of each instruction to determine setting of control points that affects the register transfer

5. Assemble the control logic

Step-0: What does a (micro)processor do?

Program is in memory, stored as a series of instructions.

Basic execution is:

1. Read one instruction
2. Decode what it wants
3. Find the operands either from registers or from memory
4. Perform the operation as dictated by the instruction
5. Write the result (if necessary)

   Go to the next instruction (and REPEAT THE CYCLE)

What are the Basic Components of MIPS?

The main pieces

- Memory to store program and data
- Registers to access data fast
- An ALU to perform the core function

Some auxiliary pieces

- A program counter to tell where to find next instruction
- Some logic to decode instructions
- A way to manipulate the program counter for branches

Let us Start with Building the Processor

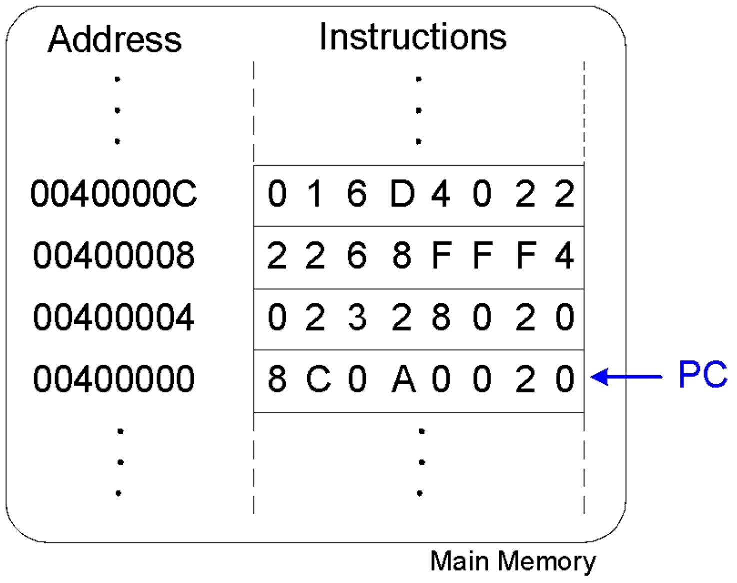
Program is stored in a memory as 32-bit binary values.

Memory addresses are also 32-bit wide, allowing (in theory) 2^{32} = 4 Gigabytes of addressed memory.

The actual address is stored in a register called PC.

Assembly Code	Machine Code
`lw   $t2, 32($0)`	0x8C0A0020
`add  $s0, $s1, $s2`	0x02328020
`addi $t0, $s3, -12`	0x2268FFF4
`sub  $t0, $t3, $t5`	0x016D4022

Stored Program



We have an Instruction, Now What ?

There are different types of instructions

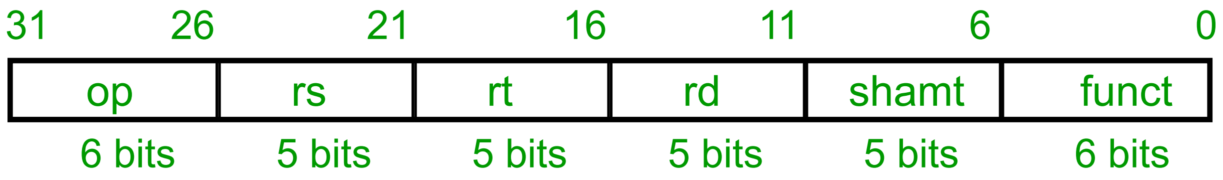
- ***R type*** (three registers)
- ***I type*** (the one with immediate values)
- ***J type*** (for changing the flow)

First only a subset of MIPS instructions:

- R type instructions: **`and`, `or`, `add`, `sub`**
- Memory instructions: **`lw`, `sw`**
- Branch instructions: **`beq`**

Later we will add `addi` and `j`

The instruction subset for today (1 or 4)



add rs, rt, rd $\longleftrightarrow$ R[rd] = R[rs] + R[rt]

sub rs, rt, rd $\longleftrightarrow$ R[rd] = R[rs] - R[rt]

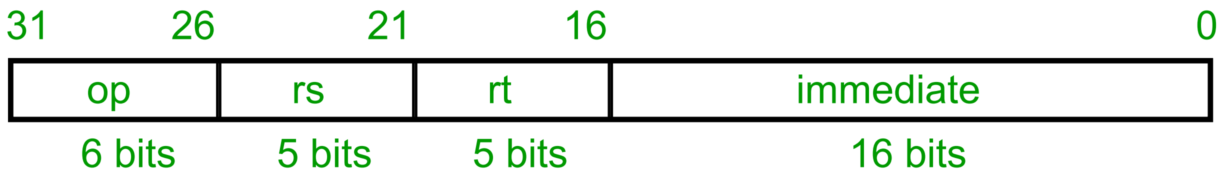
Note that we need to
1. Decode the operation
2. Get R[rs] and R[rt]
3. Add them up or subt
4. Put the result into R[**rd**]

Our register file needs at least 2 input ports

We need an ALU

Our register file needs another input port that receives sth from the ALU output

The instruction subset for today (2 or 4)



ori rs, rt, imm16 ⟷ R[rt] = R[rs] | ZeroExtImm

Note that we need to

1. Zero extend an immediate
2. Get that binary and R[rs]
3. Or them up
4. Put the result into R[**rt**]

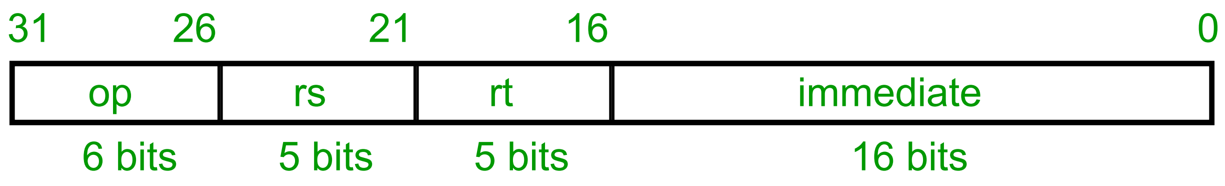
We need a special unit to achieve this extension

Our register file's one of the input ports should receive it

We should be able to use our ALU for add, sub, or, etc.

We should be able to use not only $rd and but also $rt as a target register!

The instruction subset for today (3 or 4)



lw rs, rt, imm16 $\leftrightarrow$ R[rt] = M[R[rs]+SignExtImm]

sw rs, rt, imm16 $\leftrightarrow$ M[R[rs]+SignExtImm] = R[rt]

Note that we need to

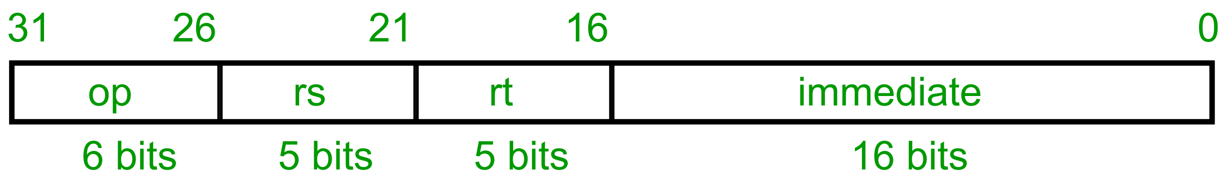
1. Sign extend an immediate
2. Get that binary and R[rs]
3. Add them up
4. Read from/write to memory

Oh! Sometimes we do sign extension (to allow go up and down in the memory)

Our register file's one of the input ports should receive it

Our ALU should be connected to the memory

The instruction Subset for today (4 or 4)



beq rs, rt, imm16 $\leftrightarrow$ if(R[rs]==R[rt])
    PC=PC+4+BranchAddr

Note that we need to

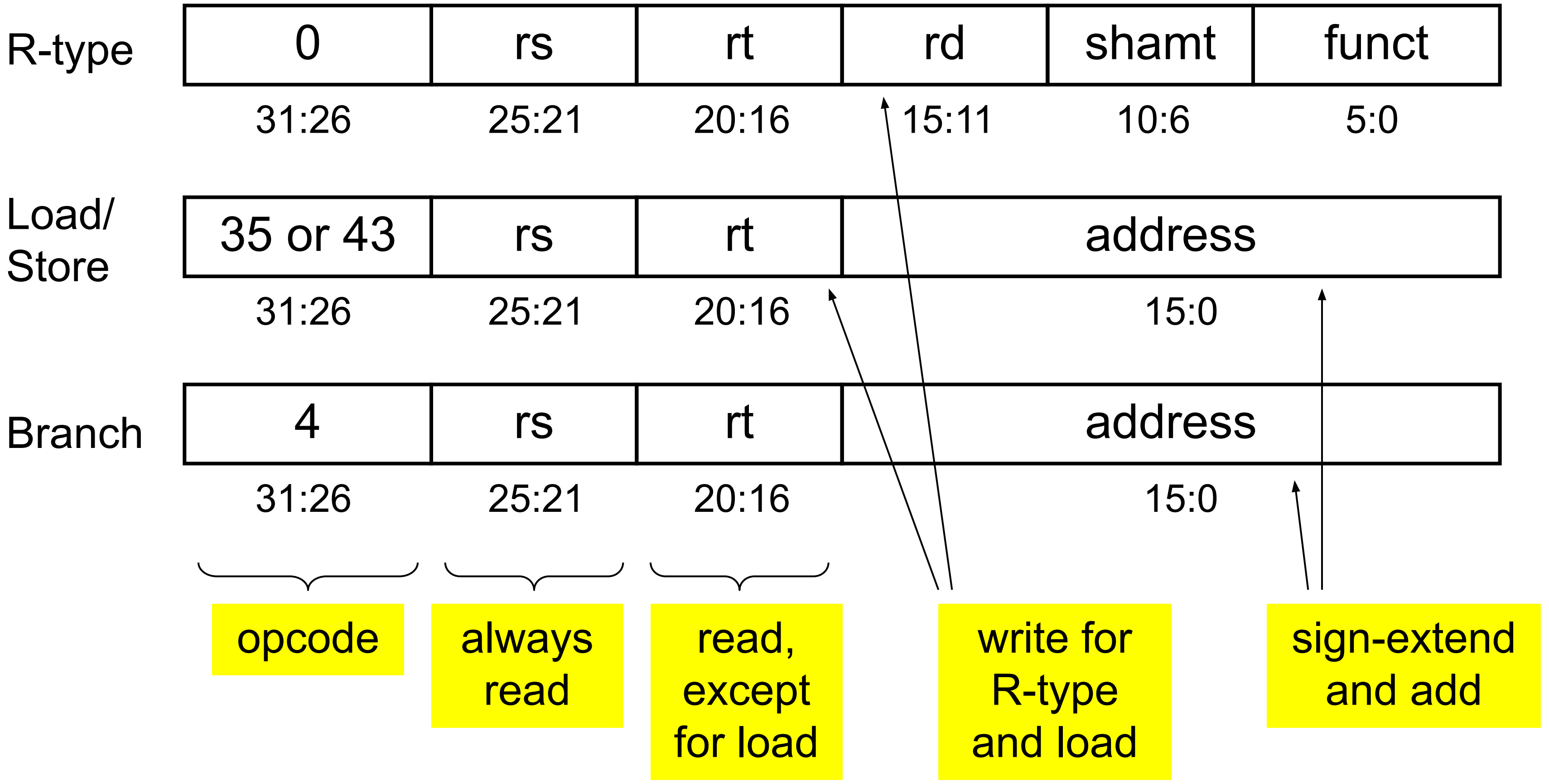
1. Get R[rs] and R[rt]
2. Compare them
3. Calculate the branch add.
4. Update the PC

Our ALU can do this, we just need to tell him/her when we need this

We need 2 left shift (because all instructions are aligned on 4-byte boundaries)

Do we need another ALU???

Big Picture



Notice that there are repeated actions with different inputs or outputs.
How can we minimize the circuitry?

Our MIPS Datapath has several options

ALU inputs

- Either RT or Immediate *(MUX)*

Write Address of Register File

- Either RD or RT *(MUX)*

Write Data In of Register File

- Either ALU out or Data Memory Out *(MUX)*

Write enable of Register File

- Not always a register write *(MUX)*

Write enable of Memory

- Only when writing to memory (sw) *(MUX)*

All these options are our control signals

And in order to implement these instructions efficiently, we will need some logic elements!!!

Logic Design Basics (Review Lecture 10!)

Information encoded in binary

- Low voltage = 0, High voltage = 1
- One wire per bit
- Multi-bit data encoded on multi-wire buses

Combinational element

- Operate on data
- Output is a function of input

State (or sequential) elements

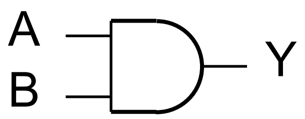
- Store information

+

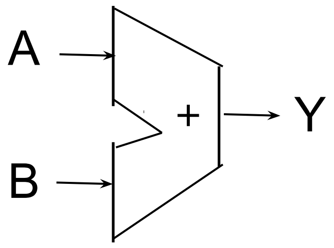
Step 2: Components of the Datapath

Combinational Elements

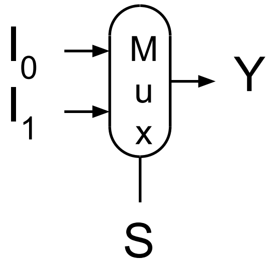
AND-gate
(Y = A & B)



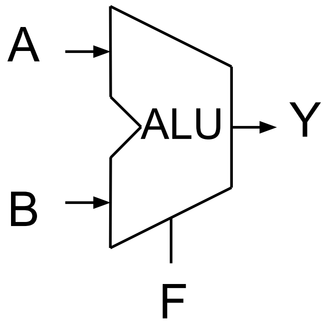
Adder
(Y = A + B)



Multiplexer
$Y = S\ ?\ I_1 : I_0$



Arithmetic/Logic Unit,
Y = F(A, B)



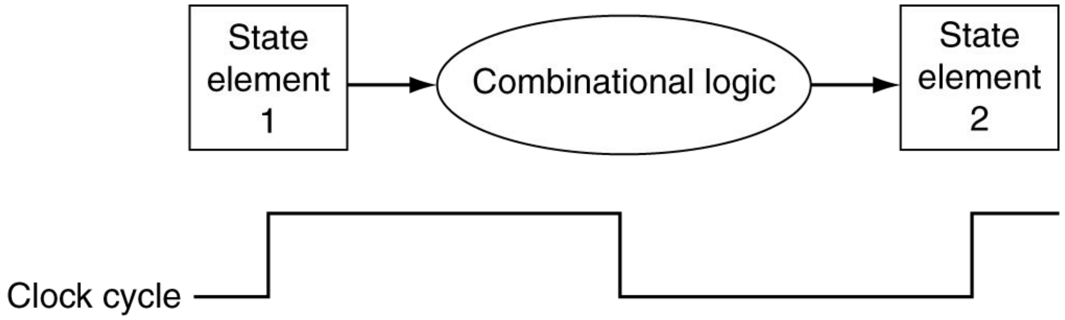
Storage Elements

- lw, sw, …
- Reading from memory or writing to memory requires a clocking!

Clocking Methodology

Combinational logic transforms data during clock cycles

- Between clock edges
- Longest delay determines clock period


State element 1
Combinational logic
State element 2
Clock cycle