Lecture 9: Processor design – multi cycle

- Aren't single cycle processors good enough? No!
- Speed: cycle time must be long enough for the most complex instruction to complete
  - – But the average instruction needs less time
- Cost: functional units (e.g. adders) cannot be re-used within one cycle
- Multiple & varied cycles per instruction means that no instruction takes more time or uses more func. units than required

Lecture outline

- Brief processor performance evaluation
- Determine the components
- Build the datapath
- Build the control

Measuring processor speed

Execution time is

instruction count

x

cycles per instruction

x

cycle time

Determine the components

Processor task

- Instruction fetch from memory
- Read registers
- Execution
  - Data processing instructions
  - Data transfer instructions
  - Branch instructions

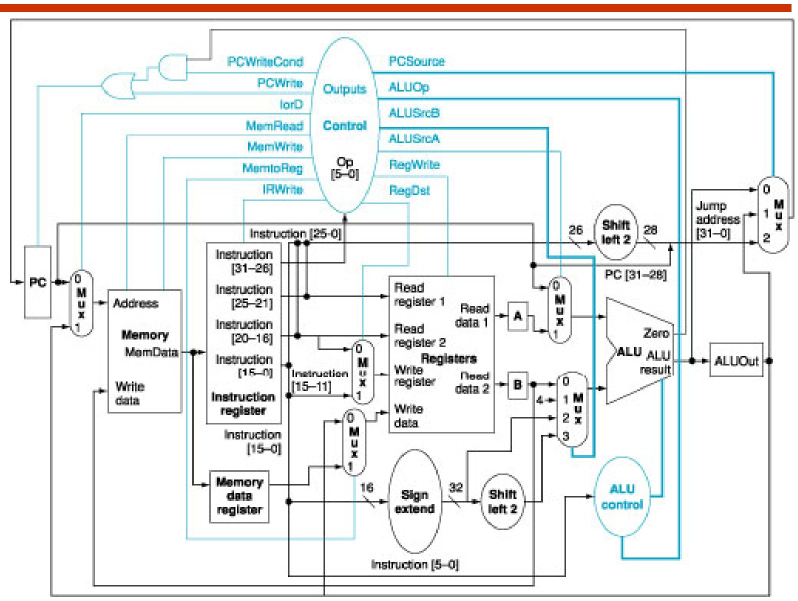
Component list

- PC register
- Memory ~~(instructions)~~
- ~~Adder: PC+4~~
- Register file
  - 2 read, 1 write
- ALU

Design guidelines

- Cycle time determined by the delay through the slowest functional unit
- Reuse functional units as much as possible
  - – Multiplexors added to select the different inputs
- At end of each cycle, data required in subsequent cycles must be stored somewhere
  - – Data for other instructions are kept in the memory, register file, or the PC
  - – Data for same instruction are kept in new registers not visible to the programmer

Multi-cycle datapath



How to design the control part

- The control unit of a multicycle processor is an FSM
- Determine exactly what happens in each cycle
- and what is the next step
- Be careful with register load-enable control signals

What happens in each cycle – 1 & 2

1. Instruction fetch

```
IR <= Mem[PC]
PC <= PC+4
```

2. Instruction decode and register fetch

```
A <= Reg[IR[25:21]]
B <= Reg[IR[20:16]]
ALUOut <= PC+sgnext(IR[15:0]<<2)
```

What happens in each cycle – 3

3a. Memory address generation

```
ALUOut <= A+sgnext(IR[15:0])
```

3b. R-type arithmetic-logical instruction

```
ALUOut <= A op B
```

3c. Branch completion

```
if (A == B) PC <= ALUOut
```

3d. Jump completion

```
PC <= {PC[31:28],IR[25:0],2'b00}
```


What happens in each cycle – 4

4a. Memory access (load)

```
MDR <= Mem[ALUOut]
```

4b. Memory access (store) & completion

```
Mem[ALUOut] <= B
```

4c. R-type arith-logical instruction completion

```
Reg[IR[15:11]] = ALUOut
```

What happens in each cycle – 5

5. Load instruction completion

```
Reg[IR[20:16]] <= MDR
```


State diagram