ECE3700J Final RC

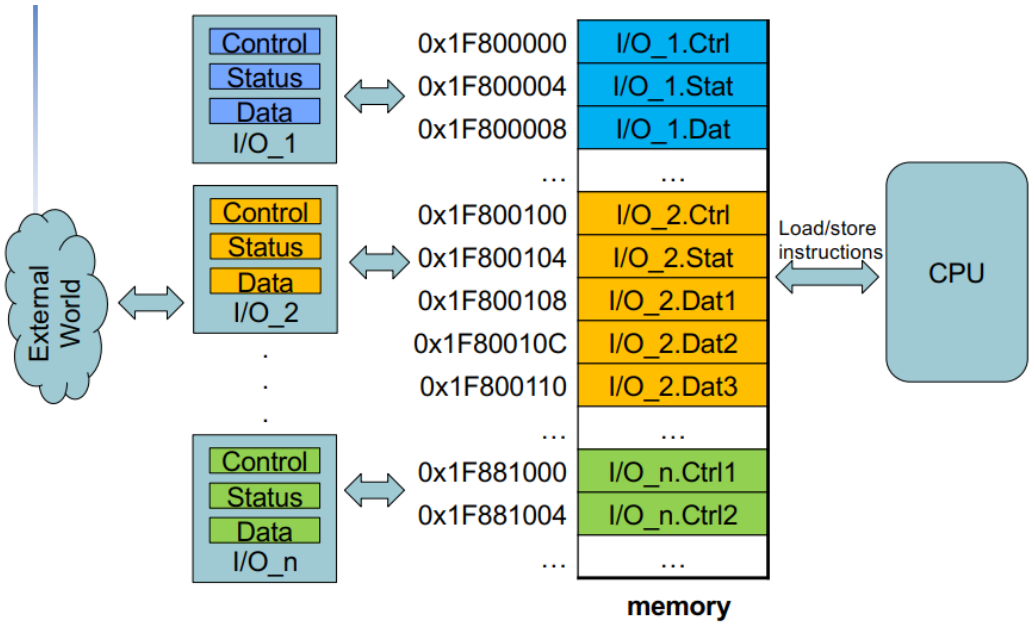
IO & Interfaces

IO Control Registers

1. control register
2. status register: indicate device status
3. data register

Memory Mapped IO

1. IO regs ↔ Memory locations
2. (Software) access IO by regular (virtual) memory address
3. (OS) operate IO by read/write on memory
4. OS uses address translation mechanism to make them only accessible in kernel mode



Communication

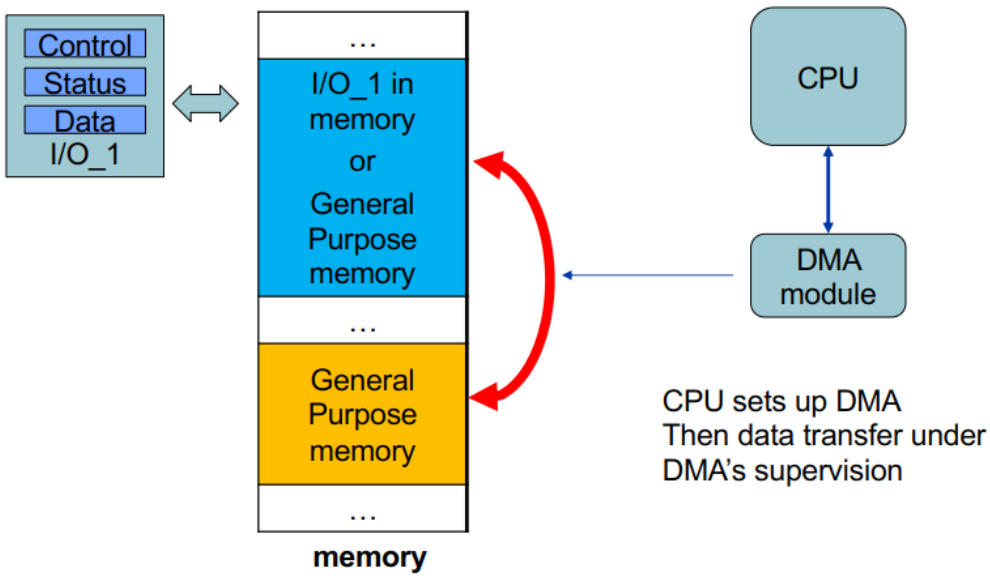
1. Polling P13
2. Interrupt P14

Direct Memory Access

1. DMA
2. memory ↔ data ↔ IO deivces

DMA Controller

1. Be set up by processor, contains necessary information including device ID, start address, number of bytes, triggering events
2. Transfer data **autonomously** without CPU
3. Be started by IO controller interrupt or CPU request data
4. Call CPU by **interrupt** on completion or critical events or error



T14 Checklist

1. What are I/O devices?

2. What is volatile / non-volatile memory? Give some examples.
3. What is LAN / WAN? What is Wi-Fi / Bluetooth? What is the connection between Wi-Fi and WLAN?
4. What are the **two** aspects when measuring the performance of I/O system?
5. Why is I/O managed by OS?
6. What are I/O control registers?
7. **What is Memory mapped I/O?**
8. What is polling? Where is it used? Good or bad?
9. What is interrupt? What is interrupt priority?
10. What is DMA?
11. What are the possible problems when DMA meet cache? How to solve them?
12. What is **bus**? Good or bad? List some bus types.

Control Hazard

3 methods:

1. stall
2. assume not
3. predict

Do not mix them up!

Stall

1. Normally, `beq` need ____ nops behind if the _____ is in MEM.
2. Move from MEM to ____, we only need 1 nops.
3. What do we add?
4. Are we done? How to **“stall nop”?** Two methods. Review HW6Q1

a. PC <= PC;

   IF_ID_reg <= nop;

b. Move branch or jump instruction detection to IF stage;

   nop_address <= nop;

Assume branch not taken

1. flush: make sth. nop
2. All 0 control signals

Exercise. Write the condition of “IF.Flush = 1”.

Exercise. Write the condition of “PC = PC + 4” if `beq` detected in ID stage.

Relevant Issues

1. performance

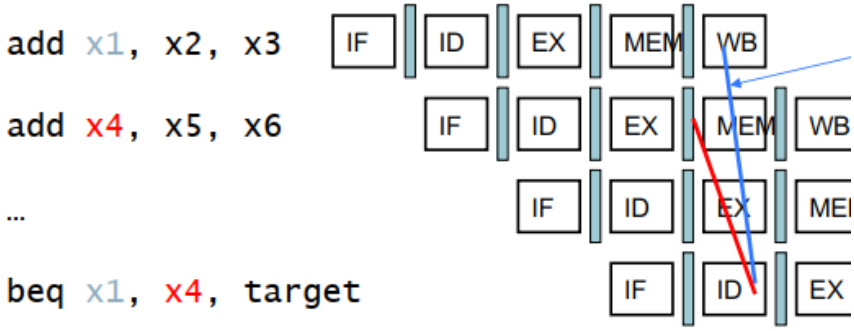
   Exercise. If we want to support `jalr` instruction in pipeline, we identify it in ID stage. However, we can choose to put calculation unit in EX stage, or move it to ID stage. Which stage is definitely longer? Which stage could be longer? Which one is better?

2. new data hazard

   a. Normally, there should be **2** instructions between `add x4 x0 x0` and `beq x0 x4 target`

      ▼ From WB to ID: not a forwarding path

      Be familiar with time stage diagram. What is forwarding path?



b. If only one → forwarding path

   Exercise. Forward1 = 1, when

c. If none, stall nop.

   Exercise. How many instructions should be in between `lw x4 ...` and `beq x4 ...` without any nop or forwarding path? What forwarding path should be used when there is not enough instructions in between?

Exercise. How to add stall in a current time stage diagram? HW6Q3

Branch Prediction

static prediction: typical: loop / if-else

dynamic prediction

1. check table (branch prediction buffer / branch history table)
   a. expect same outcome for branch instruction in same address
   b. penalty: +flip

address	instruction	1-bit prediction	Target address
0x0008F000	beq	1 (taken)	0x0008F020
0x0008F0C4	beq	0 (not taken)	0x0008F080
0x0008F310	bne	0	.
0x00090000	beq	1	.
.	.	.	.
.	.	.	
.	.	.	

2. 2-bit Predictor

Exercise. Draw the FSM for 2-bit predictor.

Exercise. Use the example below to explain why 2-bit predictor is better than 1-bit.

```
Outer: ...
       ...
Inner: ...
       ...
       beq ... Inner
       beq ... Outer
```

Exercise. Is 2-bit predictor always better than 1-bit? If so, prove it, if not make a counter example.

Homework Problems

1. **Generally, write everything as detailed as possible! This is really important!**
2. There is almost no “don’t know” or “don’t care” in this course.

3. By default, we assume the (memory) address you write is **byte address**, recall we learned in first half of this semester that memory is **byte addressable.**

   If you really want to use word address, then… (like Slides T10 cache example)

4. **Clearly** state the address/data you write is in **binary/decimal/hexadecimal**
   a. Typically, without notation, we assume digits are in **binary if only 0/1 else decimal**
5. If you want to write the value stored in address `x`, write `Mem[x]`
   a. If it is a word (more than one bytes), write `Mem[x~y]`