ECE222

Andy Zhang

Fall 2014

Contents

Chapter 1

Computers

1.1 Classes (1.1)

1.1.1 Personal

- Desktop, laptop, tablet, smartphones
- %1 of all CPUs sold(10 billion in 2008)
- Cost: $20 - 200

1.1.2 Embedded

- Integrated into a larger device or system
  - Automotive(airbags, ABS, ...)
  - Appliances: stove, microwave...
  - Airplanes
- 99% of all CPUs
- Cost: Microchip PIC12: $0.41

1.1.3 Servers

- Provides service to many users
  - Cloud computing (Amazon EC2, Azure ...)
  - Mainframes (IBM System Z) used by banks, universities, governments due to high reliability
  - Supercomputers — weather modelling, protein folding..
- <1% of all CPUs sold
- cost: $2000 / chip

1.2 Structure (1.2)

Definition : a computer is a 'programmable device that can store, retrieve and process data' — Merriam Webster

Computers of all classes can be decomposed into five types of functional events

1. Input: Mouse, Punchard, Touch Screen, Camera
2. Output: Printer, Screems.
3. Storage: Data, instructions (binary)
    - Memory is organized into a linear array of bytes
4. ALU: Arithmetic Logic Unit
    - Performs operations on data stored in registers
    - Add, multiply, AND, NOT, ...
5. Control Unit
    - Interpret instructions, fetch operands, control ALU

Chapter 2

Processors

2.1 Processor

PC program counter stores memory address of next instruction

IR instruction register stores instruction read from memory

MAR memory address to register outputs address to memory

MDR memory data register. Holds data/instructions from memory or going to memory

2.2 Instruction execution

2.2.1 Instruction Fetch (IF)

- Copy PC contents to MAR and assert R/$\bar{W}$ control signal
- Wait for response from memory and copy MDR contents to IR
- Increment PC

2.2.2 Instruction Decode (ID)

- Interpret bits in IR

2.2.3 Operand Fetch (OF)

- Read data from registers and/or extract constants from IR

2.2.4 Execute (EX)

- Use ALU or read memory (load) or write memory (store)

2.2.5 Writeback (WB)

Write result to a register

Eg Execute Load R2, LOC (memory address label)

1. Always same as above
2. Recognize 'Load'
3. Extract LOC from IR
4. Copy LOC to MAR and assert R/$\bar{W}$ control signal
5. Copy MDR Contents to R2

2.2.6 Homework

```
ADD R4, R2, R3 (R4 <- [R2] + [R3])
Store R4, LOC
```

2.3 Design Paradigms

2.3.1 CISC

Complex Instruction Set Computer

- Machine instructions can perform complex operations

  E.g. (x86) *movsb* copies an array of bytes
- Instructions are variable length
- Operands come from registers or memory

  E.g *M68K* ADD DO, LOC (mem[LOC] <- [DO] + [mem[LOC]])
- Complex addressing modes

  E.g. (M68K) ADD DO, (A0)+
- Smaller object code
- Direct support of High Level Language constructs
- Ease of assembly language programming
- Hardware is difficult to pipeline (speed up)

2.3.2 RISC

Reduced instruction-set computer

- Fewer, simpler instructions
- Load/store architecture
  - only load or store
  - ALU operands only come from registers

Eg (ARM)

```
ldr r1, LOC
add r1, r0, r1
ldr r2=LOC
str r1, [r2]
```

- Object code is larger (by ∼%30)
- Hardwire easier to pipeline

2.4 Register Transfer Notation

(no standard)

- Expresses the semantics of instruction execution as data transfers and control flow (logic)
- Memory locations are assigned labels e.g. LOC, A
- Registers are named R0, R1, PC, IR

x denotes contents of x

E.g.

- [LOC] contents of memory at LOC
- [R0] contents of register R0
- [[R0]] contents of memory at the location specified by contents of R0

',' denotes parallel
';' dnotes sequential

E.g. ADD R4, R2, R3

R4 <– [R2] + [R3]

E.g. instruction fetch

MAR <– [PC], R/$\bar{W}$ <– 1, PC <– [PC] + 4
IR <= [MOR]

2.5 Memory

- A processor can access a finite amount of physical memory, determined by the # of address pins
- Memory is measured in binary units but reported with S.I. prefixes (e.g. 1kB = 1024 bytes)
- Hard disks are measured in decimal units (e.g. 1kB = 1000 bytes)
- IEC introduced binary units to eliminate confusion (e.g. 1kiB = 1024 bytes)

Endianness

- Big endian: MSB (most sig. byte) is at low address, LSB at high address
- Little-endian: MSB at high address, LSB at low address

Ex: 1234AB**CO**

Big Endian		Little Endian	
0	12	0	CO
1	34	1	AB
2	AB	2	34
3	CO	3	12

Chapter 3

ARM

3.1 Background

- Acon/Adoned RISC Machines
- License designs to other companies to manufacture
- Target low power/low cost

3.2 Design Principles

RISC but with some CISC characteristics
RISC
- Fixed instruction size
- load/store architecture

CISC
- Autoincrement/decrement addressing modes
- Move multiple values from registers to memory, in 1 instruction
- Condition codes

3.3 Memory

- Data sizes:
  - Word = 32 bits
  - Half word = 16 bits
  - Byte = 8 bits
- Word addresses are 'word aligned' (multiple of 4)
- Little or big endian
- Loads of half words or bytes at 200 extended or sign extended to 32 bits

3.4 Registers

Registers
- All registers are 32 bits
- 13 General purpose registers R0-R12, and

R13 is the stack pointer (SP)
R14 is the link register (LR)
R15 is the program counter (PC)

- Condition code flags

R28 (V) Overflow
R29 (C) Carry out
R30 (Z) Zero
R31 (N) Negative

- Program status register

3.5 Instruction Set

3.5.1 Variations

3 variations

- ARM — 32 bit Thumb — 16 bit (compact, limited instructions, 8 operands) Thumb 2 — Mix of 16 and 32 bit, Cortex M3 in lab

3.5.2 Data Processing Instructions

Most have this format

```
<op>{flags}{cond} Rd, Rn, Op2
```

Op2 Operand 2 (right operand)
Rn Source register (Left operand)
Rd Destination register
{cond} Execute if condition is true
{flags} E.g. S => set condition code flags
< op> Operation meumonic

```
ADDEQ R2, R0, #1
if Z==1, R2 <- [R0] + 1
```

Operand 2

- an 8 bit constant (optionally rotated)
- Register value (Rm) optionally shifted

LSL Logical Shift Left, shift all bits to the left with 0 as LSB (multiply)
LSR Logical Shift Right, shift all bits to the right with 0 as MSB (unsigned divide by two)
ASR Arithmetic Shfit Right, shift all bits to the right, MSB becomes itself (signed divide by two)
RDR Rotate right
RRX Rotate right extend

```
ADD r2, r0, r1, LSL #2
r2 <- [r0] + [r1] << 2
= r2 <- [r0] + 4*[r2]
```

Arithmetic ADD, ADC (add with carry), SUB, SBC, RSB (reverse subtract)

Logical (bitwise) AND, ORR, EOR, BIC (and not, bitwise clear), ORN (or not), (no NOT) —— EOR Rd, Rn \#0xffffff