

COL216

Computer Architecture

Introduction
4th Jan, 2018

What this course is about?

PROGRAMS

- Expressions
- Types
- Conditions
- Loops
- Functions
- Classes
- Threads

Computer Architecture

CIRCUITS

- Transistors (v , i)
- Gates (1, 0)
- Flip-flops
- Registeers
- Memories
- ALUs
- FSMs

Many programming languages

C

C++

Java

Python

Matlab

Computer
Architecture

CIRCUITS

- Transistors (v, i)
- Gates (1, 0)
- Flip-flops
- Registeers
- Memories
- ALUs
- FSMs

Can you use a ‘bare’ computer?

- A computer usually comes with ‘hardware’ and ‘system software’
- What can a ‘bare computer’ or just the hardware do?
- What language does it understand?
- A ‘bare’ computer is like an unfurnished building

How a program runs on a computer?

- Compiler
 - different compilers for different programming languages
- Libraries
- Linker
- Loader
- Operating system

Software Abstraction

```
swap (int v[ ], int k);  
{ int temp;  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp;  
}
```

C

swap:

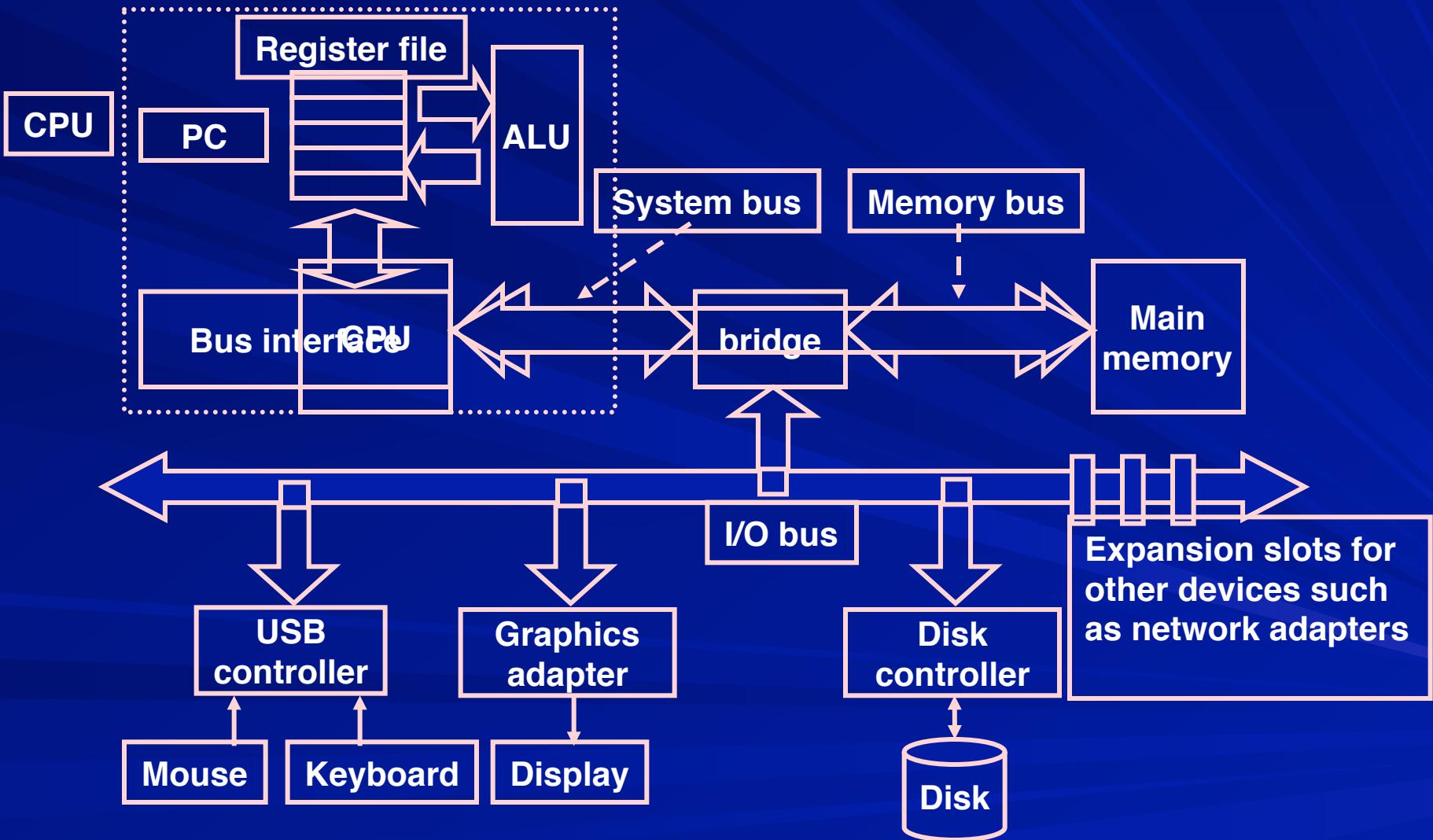
```
mul r3, r5, r4  
add r2, r3, r2  
ldr r6, [r2, #0]  
ldr r7, [r2, #4]  
str r7, [r2, #0]  
str r6, [r2, #4]  
mov pc, lr
```

assembly

machine code

00001000:	E0030495
00001004:	E0832002
00001008:	E5926000
0000100C:	E5927004
00001010:	E5827000
00001014:	E5826004
00001018:	E1A0F00E

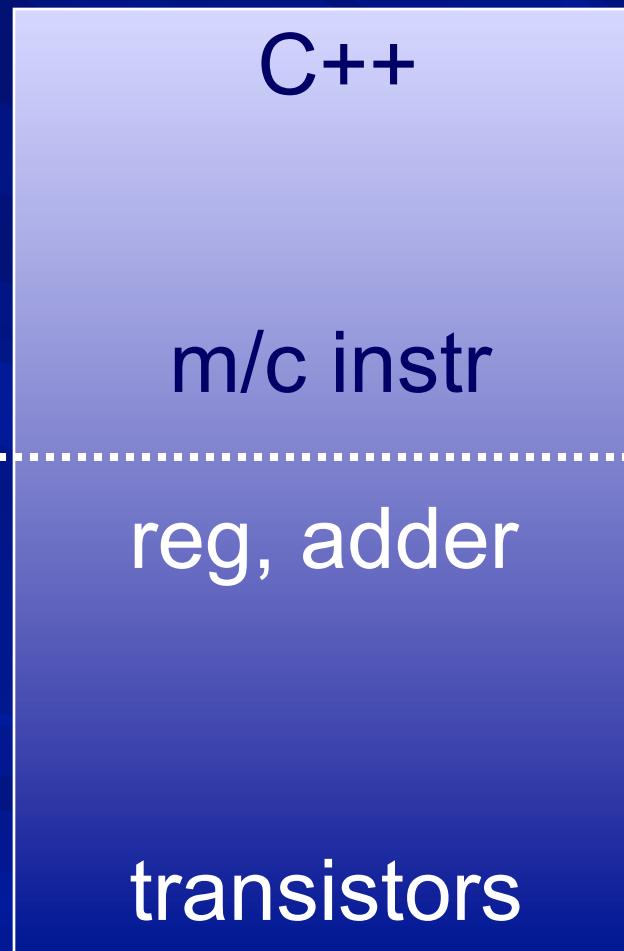
Hardware abstraction



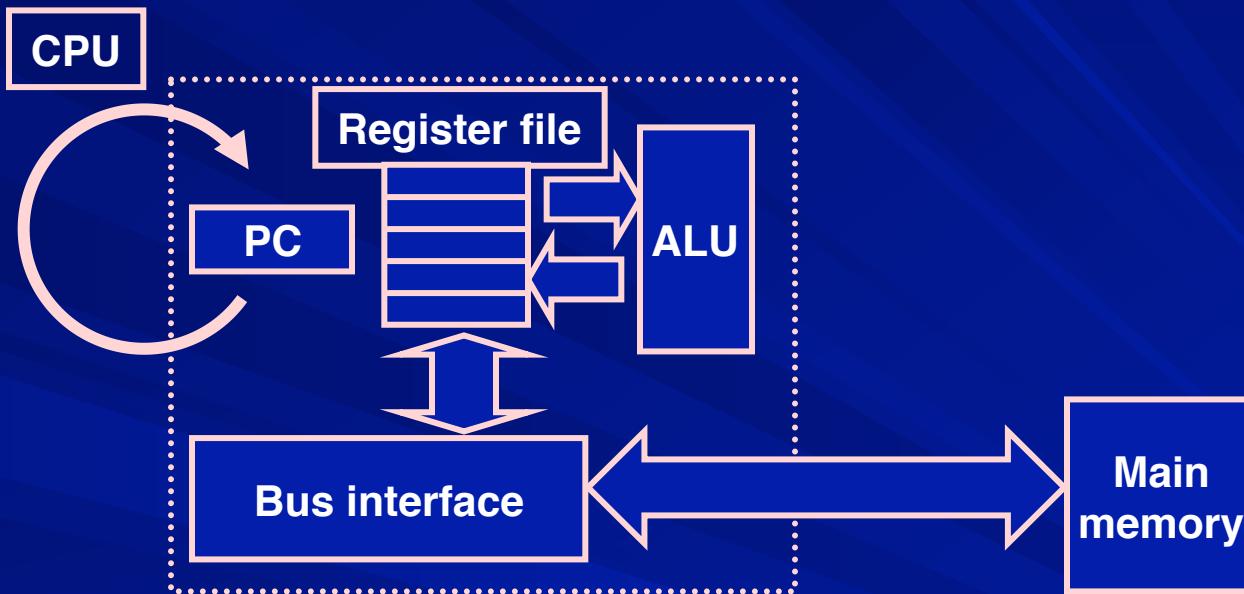
Hardware/software interface

software

hardware



Instruction execution



- Instructions for arithmetic
- Instructions to move data
- Instructions for decision making

How instructions are encoded?

Machines (and languages)

- Different types of computers
 - Desktops, laptops, servers
 - Tablets, smart phones
 - Data centres, super computers

■ Multiple manufacturers

AMD

H-P

Intel

IBM

Motorola

NVIDIA

NXP

Qualcomm

Samsung . . .

What we will study

ARM (Advanced RISC Machine)

- Most popular 32 bit ISA (Instruction set architecture)
- Used extensively in embedded systems and mobile / hand held computing devices
- Easy to understand

Arithmetic instructions

- There are 2 operands and 1 result
- The order is fixed (result destination first)

Example:

C code: $a = b + c$

ARM code: add a, b, c

actually: add r1, r2, r3

registers associated with variables by compiler

ARM arithmetic

- Simplicity favors efficient implementation
- Operands must be registers, only 16 registers provided (smaller is faster)
- Expressions need to be broken

C code

$a = b + c + d;$

$e = f - (a + b);$

ARM code

add r7, r2, r3

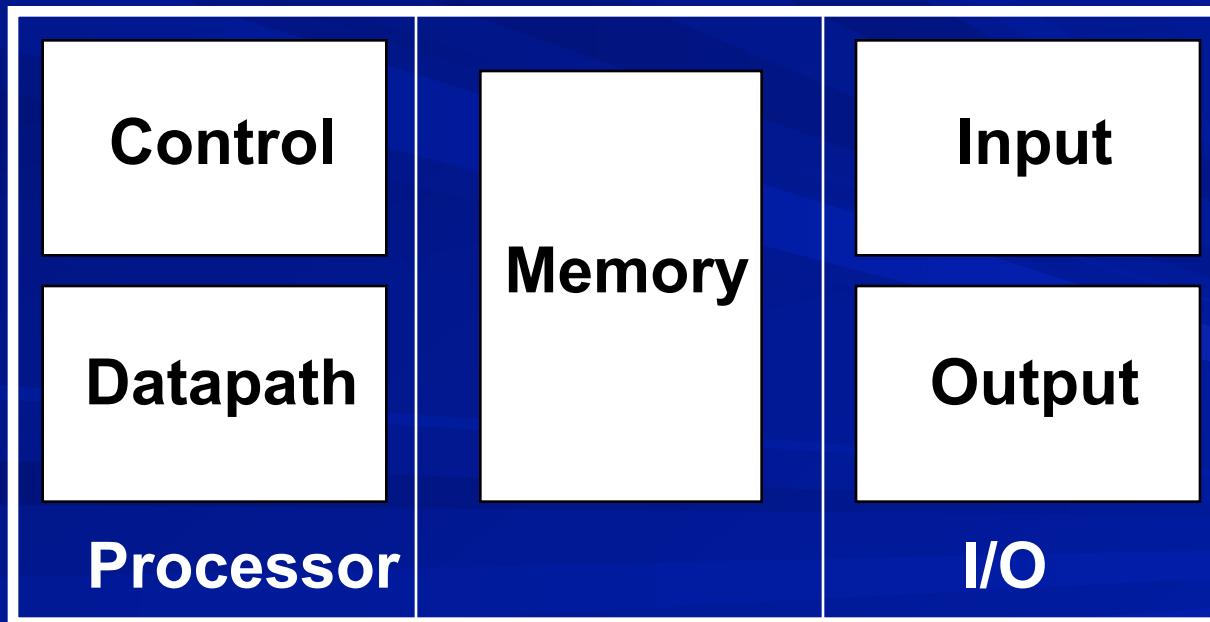
add r1, r7, r4

add r7, r1, r2

sub r5, r6, r7

Registers vs. Memory

- Scalars mapped to registers
- Structures, arrays etc in memory



Memory Organization

- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array
- "Byte addressing" means that the index points to a byte of memory.

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data
...	

Words and Bytes

- 2^{32} bytes : byte addresses from 0 to $2^{32}-1$
- 2^{30} words : byte addresses 0, 4, 8, ... $2^{32}-4$

Big endian byte order

0	1	2	3
4	5	6	7

Little endian byte order

3	2	1	0
7	6	5	4

Non-aligned word

3	2	1	0
7	6	5	4

Instructions to access memory

Load / store instructions

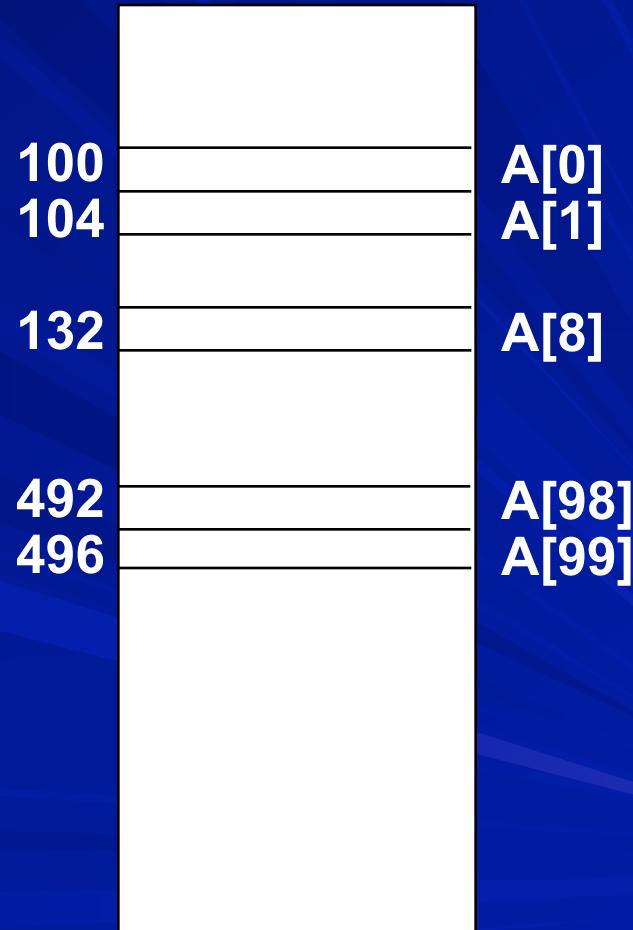
C : $v = A[8];$

ARM :

ldr r1, [r2, #32] {r2 has 100}

or

ldr r1, [r2, #100] {r2 has 32}
{not possible if address of A[0]
is large}



Load / Store example

C code: $A[8] = A[8] + h;$

ARM code:

```
ldr    r1, [r2, #32]
add    r1, r1, r3
str    r1, [r2, #32]
```

A simple example

```
swap (int v[ ], int k);
{ int temp;
  temp = v[k]
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

What does this code do?

swap:
@ r2 has addr of v[0]
@ r5 has k, r4 has #4

mul r3, r5, r4
add r2, r3, r2
ldr r6, [r2, #0]
ldr r7, [r2, #4]
str r7, [r2, #0]
str r6, [r2, #4]

return



Control

- Decision making instructions - alter the control flow
- Compare instruction : cmp
- Conditional branch instructions : beq, bne
- Example:

if (i == j)

 h = i + j;

 k = k - i;

 cmp r1, r2

 bne L

 add r3, r1, r2

L: sub r4, r4, r1

Control

- Unconditional branch instructions:

- b label

- Example:

- if (i = j)

- h = i + j;

- else

- h = i - j;

- k = k - i;

- cmp r1, r2

- bne Lab1

- add r3, r1, r2

- b Lab2

- Lab1: sub r3, r1, r2

- Lab2: sub r4, r4, r1

Other comparison operations

- blt: branch if less-than
- ble: branch if less-than-or-equal
- bgt: branch if greater-than
- bge: branch if greater-than-or-equal

Writing a simple loop for $\sum_i A[i]$

```
s = 0;  
i = 0;  
L: s = s+A[i];  
    i++;  
    if (i<n) goto L;
```

Writing a simple loop for $\sum_i A[i]$

s = 0;	mov r1, #0
i = 0;	mov r2, #0
L: s = s+A[i];	L: mul r3, r2, r7 @ r7 = 4 add r3, r3, r4 @ r4=&A[0] ldr r5, [r3, #0] add r1, r1, r5
i++;	add r2, r2, #1
if (i<n) goto L;	cmp r2, r6 @ r6 = n blt L

Improving code: pointer vs. index

s = 0;	mov r1, #0
i = 0;	mov r2, #0
p = &A[0];	mov r3, r4
L: s = s + *p;	L: ldr r5, [r3, #0] add r1, r1, r5
p++;	add r3, r3, #4
i++;	add r2, r2, #1
if (i<n) goto L;	cmp r2, r6 @ r6 = n blt L

Improving code further

s = 0;

i = 0;

p = &A[0];

L: s = s + *p;

p++;

i++;

if (i<n) goto L;

s = 0;

p = &A[0];

q = p+100;

L: s = s + *p;

p++;

if (p<q) goto L;

Improving code further

s = 0;	mov r1, #0
p = &A[0];	mov r3, r4
q = p+100;	add r6, r3, #400
L: s = s + *p;	L: ldr r5, [r3, #0]
	add r1, r1, r5
p++;	add r3, r3, #4
if (p<q) goto L;	cmp r3, r6 @ r6 = q
	blt L

Complete Assembly Program

```
.equ SWI_Exit. 0x11
.text
mov r1, #0
ldr r3, =AA
add r6, r3, #400
L: ldr r5, [r3, #0]
add r1, r1, r5
add r3, r3, #4
cmp r3, r6      @ r6 = q
blt L
swi SWI_Exit
.data
AA: .space 400
.end
```

How to execute this program?

ARMSim#

- Simulator for ARM7TDMI architecture
- Developed by University of Victoria, Canada
- Includes an assembler
- Allows step-by-step execution and breakpoints
- Displays contents of registers and memory
- Supports input-output
- A plug-in simulates a particular ARM board

Before we close,

Topics to be covered

- A specific instruction set architecture
- Arithmetic and how to build an ALU
- Constructing a processor to execute instructions
- Performance issues
- Pipelining to improve performance
- Memory: caches and virtual memory
- Input / output

Books

Primary reference:

- John L. Hennessy & David A. Patterson, "Computer Organization & Design : The Hardware / Software Interface", Morgan Kaufmann Publishers.

Other references:

- Smruti R. Sarangi, "Computer Organization and Architecture", McGraw Hill.
- Vincent P. Heuring & Harry F. Jordan, "Computer System Design and Architecture", Pearson.

Evaluation plan

■ Minor Test 1	15
■ Minor Test 2	15
■ Major Test	30
■ Quizzes / assignments	15
■ Laboratory work	25

Thank you