Outline

1. Steps in program translation
2. Hardware/Software Interface Preliminaries
   1. Instruction Set Architecture
      1. General ISA Design (Architecture)
      2. Different types of ISA: RISC vs CISC
   2. Assembly programmer’s view of the system
      1. Registers: Special and general purpose
      2. Assembly and machine code (program translation detail)
      3. Layout of ARM instructions in memory
   3. Steps in program execution
   4. Basic Types of ARM Assembly Instructions
Compile time: What does gcc do?

```
% gcc hello.c
```

“Source”

Program in C

```c
#include <stdio.h>
void func1(int a, char *b)
{
    if(a > 0)
    {
        *b = ‘a’ ;
    }
}
int main()
{
    ....
    func1();
    printf("abc");
}
```

```
“Executable”:
Equivalent program in machine language
```

0000 1001 1100 0110
1010 1111 0101 1000
1010 1111 0101 1000
0000 1001 1100 0110
1100 0110 1010 1111
0101 1000 0000 1001
0101 1000 0000 1001
1100 0110 1010 1111
Steps in gcc

- The translation is actually done in a number of steps.

```
gcc -S hello.c

gcc -c hello.c

hello.s

hello.o

Attached code in assembly

Linker (ld)

Assembler (as)
```

```latex
\texttt{gcc}
```

```
hello.s

hello.o
```

```
\texttt{a.out}
```

```
\texttt{other.o math.o}
```
Steps in gcc

- Ask compiler to show temporary files:
  % gcc -S hello.c (gives hello.s – assembly code)
  % gcc -c hello.c (gives hello.o – object module)
  % gcc -o prog_hello hello.c (gives prog_hello.o - named executable)
Include code written by others

- Code written by others (libraries) can be included
- `ld` (linkage editor) merges one or more object files with the relevant libraries to produce a single executable

Libraries can be linked statically or dynamically

Library files e.g. `math.o`: the math library
Assembly Language

A. Is the binary representation of a program.

B. A symbolic representation of machine instructions

C. A set of instructions that the machine can directly execute in case of machine
Assembly Language

A. Is the binary representation of a program.

B. A symbolic representation of machine instructions

C. A set of instructions that the machine can directly execute
Machine vs Assembly Language

- Machine Language: A particular set of instructions that the CPU can directly execute – but these are ones and zeros
- Assembly language is a symbolic version of the equivalent machine language
  - each statement (called an Instruction), executes exactly one of a short list of simple commands
  - Unlike in C (and most other High Level Languages), each line of assembly code contains at most 1 instruction
  - Instructions are related to operations (e.g. =, +, -, *) in C or Java
What is an Instruction Set Architecture?

1. Everything about h/w that is visible to the s/w and can be manipulated by it via basic machine instructions (System state)

Bank of registers:
Small and fast storage location

CPU

Memory

s/w code in assembly

hw process

How long instruction [size in bytes]

How many registers

UCSD
What is an Instruction Set Architecture?

1. Everything about h/w that is visible to the s/w and can be manipulated by it via basic machine instructions (System state)
   Example: Registers: How many? What size?
   Memory: How to access contents?

2. The set of basic machine instructions:
   
   A. What they are
   B. How they change the system state
   C. How they are encoded in binary

Bank of registers:
Small and fast storage location
Is the ISA different for different CPUs?

- Different CPUs implement different sets of instructions.
  - Examples: ARM, Intel x86, IBM/Motorola PowerPC (Macintosh), MIPS, Intel IA32 …

- Two styles of CPU design:
  - RISC (Reduced Instruction Set Computing)
  - CISC (Complex Instruction Set Computing)
RISC versus CISC (Historically)

- Complex Instruction Set Computing e.g. x86
  - Larger instruction set
  - More complicated instructions built into hardware
  - Variable length
  - Multiple clock cycles per instruction

- Reduced Instruction Set Computing e.g. ARM
  - Small, highly optimized set of instructions
  - Memory accesses are specific instructions
  - One instruction per clock cycle
  - Instructions are of the same size and fixed format
A = A*B

RISC

LOAD A, eax
LOAD B, ebx
PROD eax, ebx
STORE ebx, A

CISC

MULT B, A
Translations

- **High-level language program (in C)**

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

- **Assembly language program (for MIPS)**

```assembly
swap:  sll  $2,  $5,  2
      add  $2,  $4,  $2
      lw    $15,  0($2)
      lw    $16,  4($2)
      sw    $16,  0($2)
      sw    $15,  4($2)
      jr    $31
```

- **Machine (object, binary) code (for MIPS)**

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
...
```
The Assembly Programmer’s View of the machine

- Registers: (Very) Small amount of memory inside the CPU
- Each ARM register is 32 bits wide
  - Groups of 32 bits called a *word* in ARM
The ARM Register Set

### General Purpose Registers
- r0
- r1
- r2
- r3
- r4
- r5
- r6
- r7
- r8
- r9
- r10
- r11
- r12
- r13 (sp)
- r14 (lr)
- r15 (pc)
- cpsr

### Special Purpose Registers
- Stack Pointer
ARM Assembly Variables: Registers

- Unlike HLL like C or Java, assembly cannot use variables
  - Why not? Keep Hardware Simple
- Data is put into a register before it is used for arithmetic, tested, etc.
- Manipulated data is then stored back in main memory.
- Benefit: Since registers are directly in hardware, they are very fast
In C (and most High Level Languages) variables declared first and given a type

- Example:
  ```
  int fahr, celsius;
  char a, b, c, d, e;
  ```

- Each variable can ONLY represent a value of the type it was declared as (cannot mix and match `int` and `char` variables).

- In Assembly Language, the registers have no type; operation determines how register contents are treated.
Layout of instructions in memory

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

```
0x4000 Swap:    mov    r2, r5
0x4004          add    r4, r2, #1
0x4008          ldr    r10, [r6,r2]
0x400c          ldr    r11, [r6,r4]
0x4010          mov    r1, r10
0x4014          str    r11,[r6,r2]
0x4018          str    r1, [r6,r4]
0x401c          bx     lr
```
Machine Instructions

A. Can be thought of as numbers.

B. Are commands the computer performs.

C. Were what people originally wrote programs using.

D. All of the above.
Machine Instructions

A. Can be thought of as numbers.

B. Are commands the computer performs.

C. Were what people originally wrote programs using.

D. All of the above.
Steps in program execution

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4000</td>
<td>Swap: mov r2, r5</td>
<td></td>
</tr>
<tr>
<td>0x4004</td>
<td>add r4, r2, #1</td>
<td></td>
</tr>
<tr>
<td>0x4008</td>
<td>ldr r10, [r6,r2]</td>
<td></td>
</tr>
<tr>
<td>0x400c</td>
<td>ldr r11, [r6,r4]</td>
<td></td>
</tr>
<tr>
<td>0x4010</td>
<td>mov r1, r10</td>
<td></td>
</tr>
<tr>
<td>0x4014</td>
<td>str r11, [r6,r2]</td>
<td></td>
</tr>
<tr>
<td>0x4018</td>
<td>str r1, [r6,r4]</td>
<td></td>
</tr>
<tr>
<td>0x401c</td>
<td>bx lr</td>
<td></td>
</tr>
</tbody>
</table>
Assembly Language

A. Is in binary.

B. Allows languages to be designed for their specific uses.

C. Has one line for every machine language instruction.
Assembly Language

A. Is in binary.

B. Allows languages to be designed for their specific uses.

C. Has one line for every machine language instruction.
Basic Types of Instructions

1. Arithmetic: Only processor and registers involved
   1. compute the sum (or difference) of two registers, store the result in a register
   2. move the contents of one register to another

2. Data Transfer Instructions: Interacts with memory
   1. load a word from memory into a register
   2. store the contents of a register into a memory word

3. Control Transfer Instructions: Change flow of execution
   1. jump to another instruction
   2. conditional jump (e.g., branch if registeri == 0)
   3. jump to a subroutine