CPU instructions
Structure of a CPU

Computer system

CPU
- Instruction register (IR)
- Program counter (PC)
- Processor status register (PSR)

Arithmetic Logic Unit (ALU)

General purpose registers
- R0
- R1
- R2
- R3

Addresses
- 0 0
- 1 4
- 2 1
- 3 22

memory (RAM)

One memory cell
Components of a CPU

• General purpose registers:
  – Register is a *synonym* for *memory* in *computer science*
  – A general purpose register is a *memory cell*
  – Each general purpose register has a *unique name*
  – It is used to *store (and recall) intermediate result* of complex computations

• Arithmetic and Logic Unit (ALU):
  – Is a *complex electrical circuit* that can perform *Mathematical* (+, -, ×, /) and *logical operations* (<, ≤, ≥, and, or)
  – The output (result) of the computation (obtained by the ALU) is often stored in a *general purpose register*
Structure of a CPU (cont.)

- **Instruction register (IR):**
  - Contains the *current instruction* being executed by the CPU
  - The CPU will perform the operation indicated by the *instruction code* contained in the *instruction register*

- **Program counter (PC):**
  - The program counter is a *register* (memory cell)!
  - This register contains the *address (location in memory)* of the *next instruction* after the CPU finishes executing the *current instruction* in the instruction register
  - The *value* in the program counter will be *increased* after the CPU finishes executing *one instruction*
Processor status register (PSR):

- This register contains the **various information** about the CPU.
- Among the information contained in the PSR is:
  - The **result** of a comparison operation.
- When the CPU **compares** 2 numbers \( a \) and \( b \), the **outcome** of the comparison is stored in the PSR. The outcome of a compare operation will allow the CPU to determine the following fact between \( a \) and \( b \):
  - equal
  - not equal
  - less than
  - less than or equal
  - greater than
  - greater than or equal
The following figures illustrates the CPU's execution cycle (that goes on indefinitely):

1. The CPU first sends a request to retrieve (recall) the data stored at memory location given by the program counter (PC) (in the figure, the value of the PC = 1):

The CPU will cause the memory to retrieve (recall) the value stored at memory address (location) 1
2. In response, the **RAM memory** sends the value stored in memory location 1 (= the number 4) to the CPU which stores it in the instruction register (IR):

This number (4) will be interpreted as a code for a **computer instruction**.

The **old instruction code** (0) will be **over-written** by the **new instruction code**.
3. The program counter is then increased:

Notice that the CPU is now ready to fetch the next instruction when the current instruction (stored in the instruction register) is processed.
4. The CPU will now **execute** the instruction in the **program counter** (instruction code = **4**)

When the **CPU** is finished, it will **repeat** these steps again (and again, until you turn the computer off)....
In Computer Science, a **pointer** always *points* to a location in memory.

Examples:

- Pointer to memory location 1
- Pointer to memory location 4
Pointers: *pointing* to a location in memory (cont.)

- The computer does not have "*pointing finger*" that point to some location in memory.

- The *pointer* is represented in the computer by the *value of the address (location)* of the memory where the pointer is pointing at.
Pointers: *pointing* to a location in memory (cont.)

- **Examples:**
  - Storing the value 1 in some register/memory cell will remember the memory location 1
  - The value 1 in the register/memory cell is *effectively* pointing to the memory location 1
  - Storing the value 4 in some register/memory cell will remember the memory location 4
  - The value 4 in the register/memory cell is *effectively* pointing to the memory location 4
Example of a pointer: the *program counter* (PC)

- The *program counter* (PC) is in fact a *pointer*

Example

When the *program counter* (PC) contains the value 1:

- The *program counter* (PC) is in fact *pointing* to the memory location 1:
Program flow

• Definition: **program flow**

  • **Program flow** = the *sequence of instructions* from the program executed by the CPU

• Default program flow:

  • After executing a *non-branching* (or *non-jumping*) instruction at memory location *n*, then *next instruction* that is executed is the *instruction* at memory location *n+1*
Program flow (cont.)

• Example:

<table>
<thead>
<tr>
<th>Memory</th>
<th>0</th>
<th>4</th>
<th>1</th>
<th>22</th>
<th>56</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;add&quot;</td>
<td>&quot;compare&quot;</td>
<td>&quot;subtract&quot;</td>
<td>&quot;logical and&quot;</td>
<td>&quot;negate&quot;</td>
<td>&quot;add&quot;</td>
</tr>
</tbody>
</table>

• All of the instructions in the figure are non-branching

• Therefore, the sequence of instructions executed by the CPU starting at memory location 0 is:

1. 0 ("add")  
2. 4 ("compare")  
3. 1 ("subtract")  
4. And so on (in sequence)
Branching: changing the program flow

- Branch instruction:
  
  - A branch instruction can alter (change) the default program flow
  
  - When the CPU executes a branch x instruction, the next instruction that will be executed by the CPU is the instruction at memory location x
Branching: changing the program flow (cont.)

• Example:

<table>
<thead>
<tr>
<th>Memory</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;add&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;branch to location 4&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;subtract&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;logical and&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;negate&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;add&quot;</td>
</tr>
</tbody>
</table>

• Notice there is a branch to location 4 instruction in the example.

• Therefore, the sequence of instructions executed by the CPU starting at memory location 0 is:

1. 0 ("add")
2. 64 ("branch to location 4")
3. 56 ("negate" --- this is the instruction at memory location 4)
4. 0 ("add")
5. And so on (in sequence again)
The *types* of instructions that a computer can execute

- The **computer** is a computing (reckoning) machine
- All existing computers (actually the CPU) execute the following **3 types** of instructions:

<table>
<thead>
<tr>
<th>Arithmetic and logic operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
</tr>
<tr>
<td>−</td>
</tr>
<tr>
<td>×</td>
</tr>
<tr>
<td>/</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>OR</td>
</tr>
<tr>
<td>NOT</td>
</tr>
</tbody>
</table>

The result of an arithmetic and logic operation is often stored in a **general purpose register**
The *types* of instructions that a computer can execute (cont.)

- Memory transfer operations
  - Transfer the *content from some specific memory location* to a specific register (memory cell) in the CPU.
  - and vice versa (transfer the *content from some specific register (memory cell)* in the CPU to some specific memory location).
The *types* of instructions that a computer can execute (cont.)

- **Branch operations**
  - A *branch instruction* will cause the CPU to *branch (jump)* to the specified location in memory.
  - After the *jump* has occurred, the CPU will continue to execute the instructions *in sequence*, until another *branch/jump instruction* is encountered.
  - There are *2 kinds* of *branch operations*:
    1. A *unconditional branch* instruction will *always* cause the CPU to *jump* to the target location.
    2. A *conditional branch* instruction will *only* cause the CPU to *jump* to the target location when the *specified condition* is met.
Computer Algorithms
Fact: computers are *dumb* machines

- Basic property of a computer (a machine):
  - Computers do *what we tell* them to do
  - Unfortunately, computer *do not necessarily* do *what we want* them to do....

(Because we can *make mistake* in *telling the computer* what we *want to do*... These *mistakes* are called "*bugs""
Algorithm

• Definition: **algorithm**

Dictionary definition:

• **Algorithm** = a step-by-step procedure for solving a problem or accomplishing some task, especially by means of a **computer**
Computer Algorithms

• Computer Algorithm:

is an algorithm that can be executed by a computer
Properties of computer algorithms:

- The steps in an algorithm must be consists of operations that can be executed by a computer.
- The step in an algorithm must be unambiguous.

(remember that a dumb machine like a computer will do what it is told to do.)

Resolving ambiguity requires some thinking (intelligence) which computers cannot do. !)
A real life algorithm

• Here is a famous algorithm that you can try out:

  • Take 2 pieces of paper and write on one piece "A" and on the other piece "B":

  A  B

  • Write a positive integer on each piece of paper.
Euclid's Algorithm:

As long as one of the number is not zero (0) do

{  
    if ( number on A ≥ number on B )
        replace the number on A by the value (A - B)
    otherwise
        replace the number on B by the value (B - A)
}

The Greatest Common Divisor (GCD) of the numbers A and B is the non-zero number on one of the papers.
A real life algorithm (cont.)

- Example of the Euclid Algorithm:

- We start with the following 2 numbers:

- Since number on A < number on B, we replace the number on B (= 36) by the value \((B - A) = 8\):
A real life algorithm (cont.)

• Now number on A ≥ number on B, so we replace the number on A (= 28) by the value \((A - B) = 20\):

\[
\begin{array}{cc}
A & B \\
20 & 8 \\
\end{array}
\]

• Again, number on A ≥ number on B, so we replace the number on A (= 20) by the value \((A - B) = 12\):

\[
\begin{array}{cc}
A & B \\
12 & 8 \\
\end{array}
\]
A real life algorithm (cont.)

• Yet again, number on A ≥ number on B, so we replace the number on A (= 12) by the value (A − B) = 4:

• This time, number on B > number on A, so we replace the number on B (= 8) by the value (B − A) = 4:
A real life algorithm (cont.)

• Now, number on A ≥ number on B, so we replace the number on A (= 4) by the value (B − A) = 0:

• One of the number on the papers is now zero

According to the Euclid's Algorithm, the other number (= 4) is equal to the GCD of (28, 36)
Programming a computer
What does *programming a computer* mean?

• Programming a computer:

  • Programming a computer = instruct a computer to perform a task/solve a problem

• Since a computer can only execute machine instructions (encoded in binary numbers), this means:

  • Write the steps of an *algorithm* using machine instructions !!!
**High level programming language**

- A high level programming language consists of (English-like) "people" language to *simplify* the writing computer algorithms.
- A high level programming language allows the programmer to write *sentences* in this language which can be *easily translated* into *machine instructions*.
- The *sentences* written in a high level programming language are called:
  - *statements*
A taste of computer algorithm

• Recall the Euclid's Algorithm: (to find the GCD)

As long as one of the number is not zero (0) do 
{ 
  if ( number on A ≥ number on B ) 
    replace the number on A by the value (A - B) 
  otherwise 
    replace the number on B by the value (B - A) 
} 

The Greatest Common Divisor (GCD) of the numbers A and B is the non-zero number on one of the papers
A taste of computer algorithm (cont.)

Here is the **Euclid Algorithm** written in the **Java programming language**:

```java
public class Euclid {
    public static void main(String args[]) {
        int A; // Memory cell named "A"
        int B; // Memory cell named "B"
        // These memory cells are like the 2 pieces of paper
        // we used above. They can store and recall a value
        A = 28; // Write "28" on the piece of paper named "A"
        B = 36; // Write "36" on the piece of paper named "B"
        System.out.println("A = " + A + " B = " + B); // Print A and B
        while (A != 0 && B != 0) {
            if (A >= B) {
                A = A - B; // Replace the number on A by (A-B)
            } else {
                B = B - A; // Replace the number on B by (B-A)
            }
            System.out.println("A = " + A + " B = " + B); // Print new A and B
        }
        System.out.println("GCD = " + B);
    }
}
```
A taste of computer algorithm (cont.)

- **Example Program:** (Demo previous code)
  
  http://mathcs.emory.edu/~cheung/Courses/170/Syllabus/01/Progs/Euclid.java

How to use this program:

- Right click on the link and save in a scratch directory
- Compile with: `javac Euclid.java`
- Run with: `java Euclid`
Computer Programs

• Computer Program:

  - Computer program = a computer algorithm that is expressed (written) in some programming language
  - Most programming languages are “English-like”

They use words from the English language
The compilation (= translation) process (cont.)

• Before the computer can execute the instructions in a C program, we must first compile (translate) it into machine operations using a C translator (compiler):
  • `cc prog.c`

• `cc` is the command to run the C compiler (translator)
• The second word `prog.c` is the name of the file that contains the C program (that is being translated)
• The machine operations that result from the translation process is stored in the file:
  • `a.out`
The compilation (= translation) process (cont.)

• Now we can **execute** the program in **machine instruction** by using the command:

  ```
  >> a.out
  ```

• The following figure summarizes the **translation process**:

![Diagram showing the compilation process from C program to machine instructions]
The compilation (= translation) process (cont.)

• Note:

- A **C compiler** is a program that is **executed** by a **computer**
- Therefore, a **C compiler** consists of **machine instructions**!

(Computers only execute machine instructions !)
Writing computer programs: trust your compiler! (cont.)

• Therefore:

• We will **not dwell** on **machine code** any longer in this course
• We will learn to **write computer programs** in a **high level programming language**

• The **high level programming language** used in this course is: