Lecture 7: Instruction Set Architecture

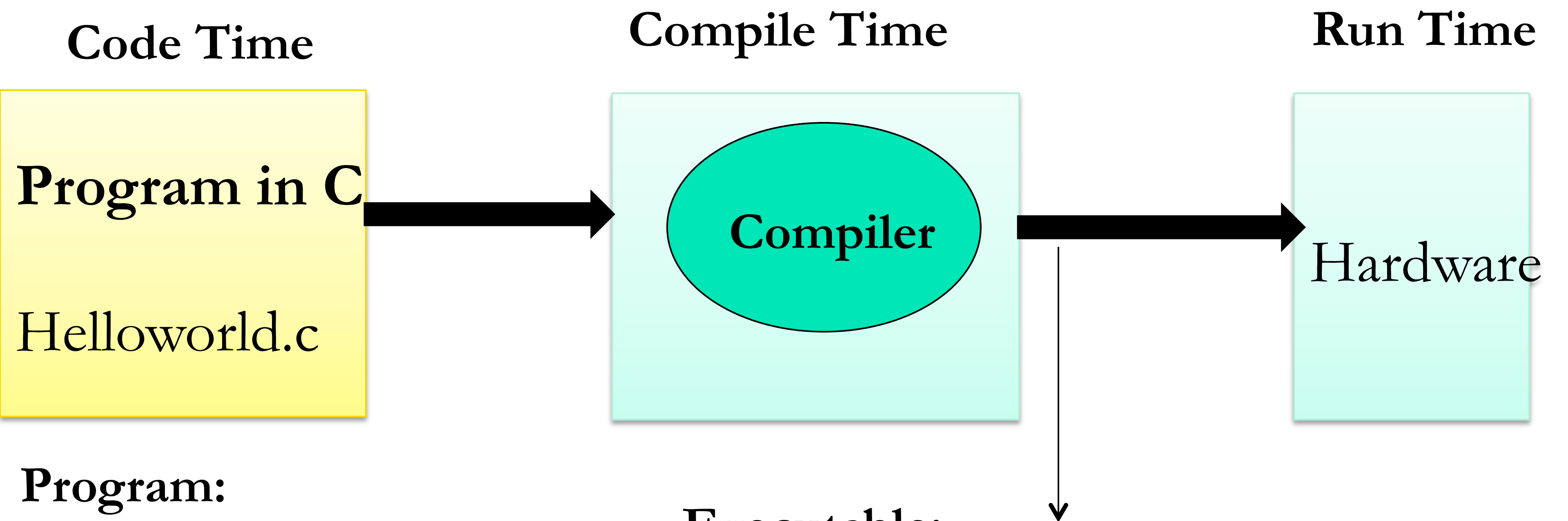
CSE 30: Computer Organization and Systems Programming
Winter 2014

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego

Outline

1. Steps in program translation
2. Hardware/Software Interface Preliminaries
    1. Instruction Set Architecture
        1. General ISA Design (Architecture)
        2. Architecture vs. Micro architecture
        3. 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

Steps in program translation



Program:
Text file stored on computers hard disk or some secondary storage

Executable:
Program in machine code
+Data in binary

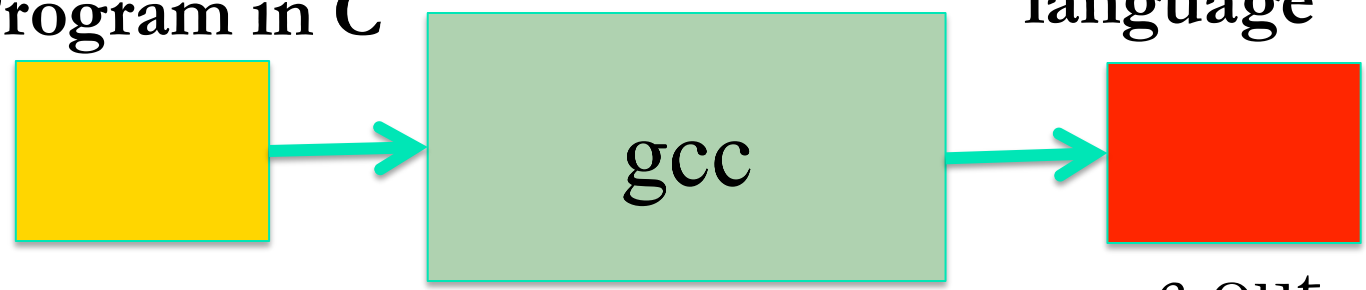
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100

Compile time: What does gcc do?

% gcc hello.c

"Executable":
Equivalent program in machine language

"Source"
Program in C



hello.c

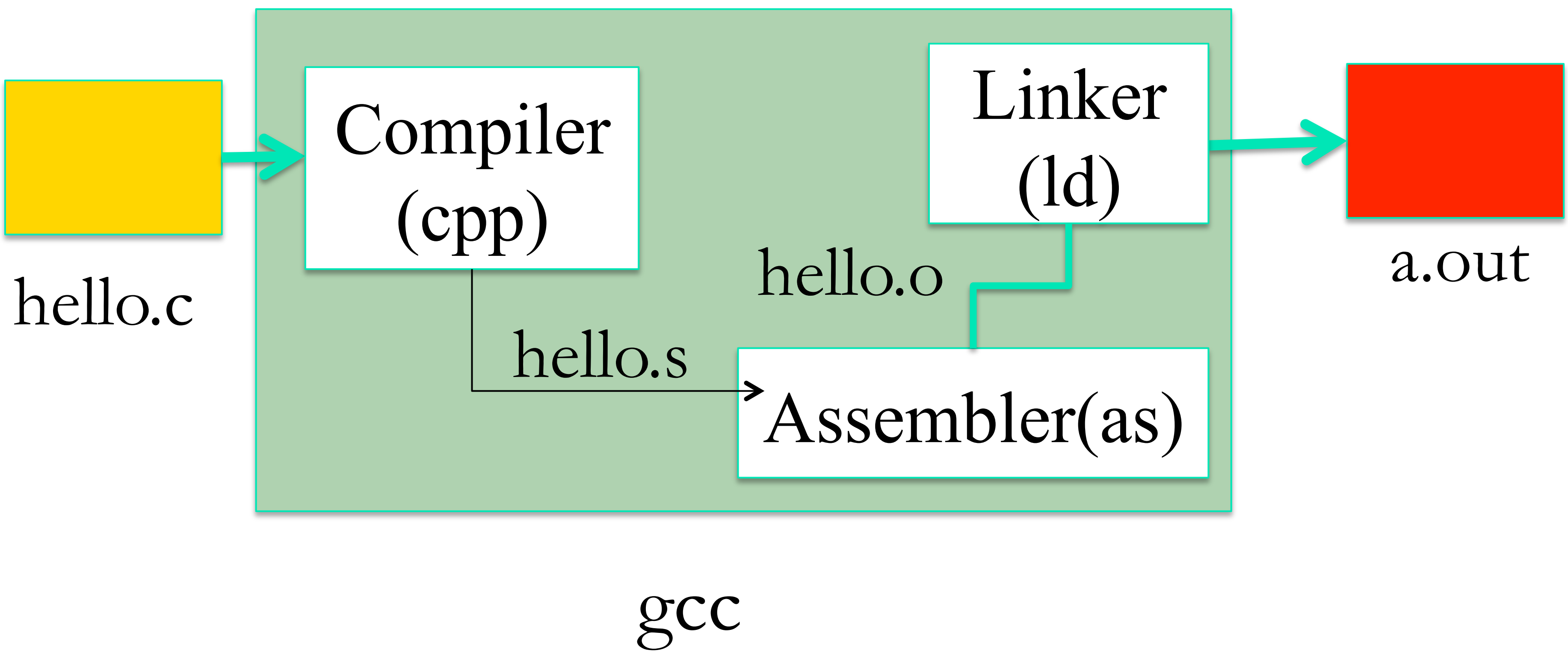
a.out

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

```
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



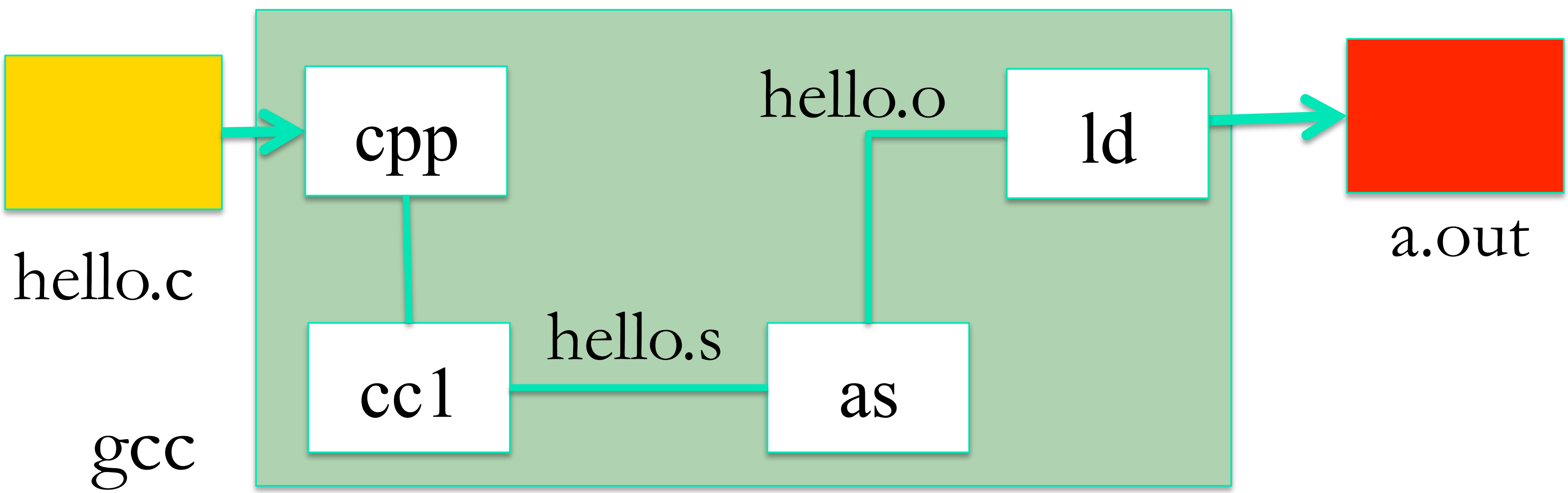
Steps in gcc

- Ask compiler to show temporary files:

% gcc –S hello.c (gives hello.s – assembly code)

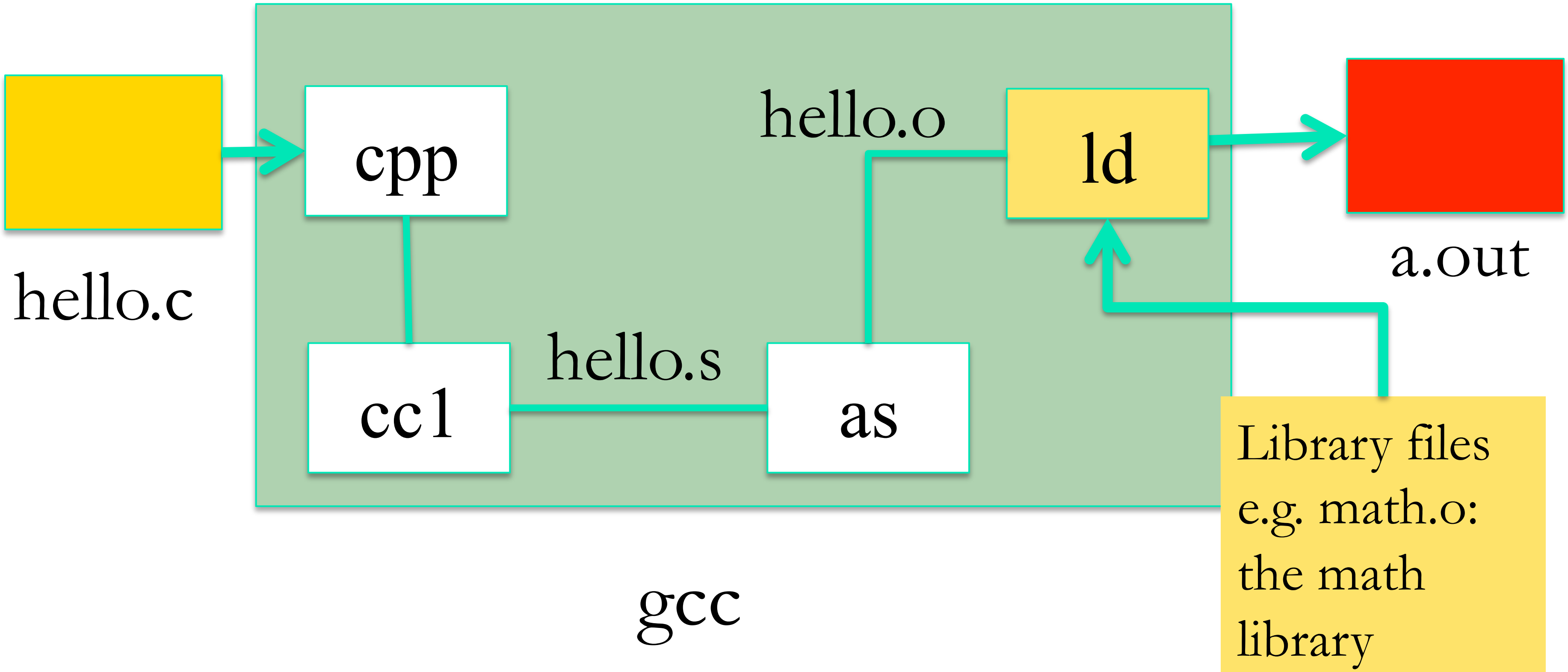
% gcc –c hello.c (gives hello.o – object module)

% 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



UCSD

Assembly Language

A. Is the binary representation of a program.

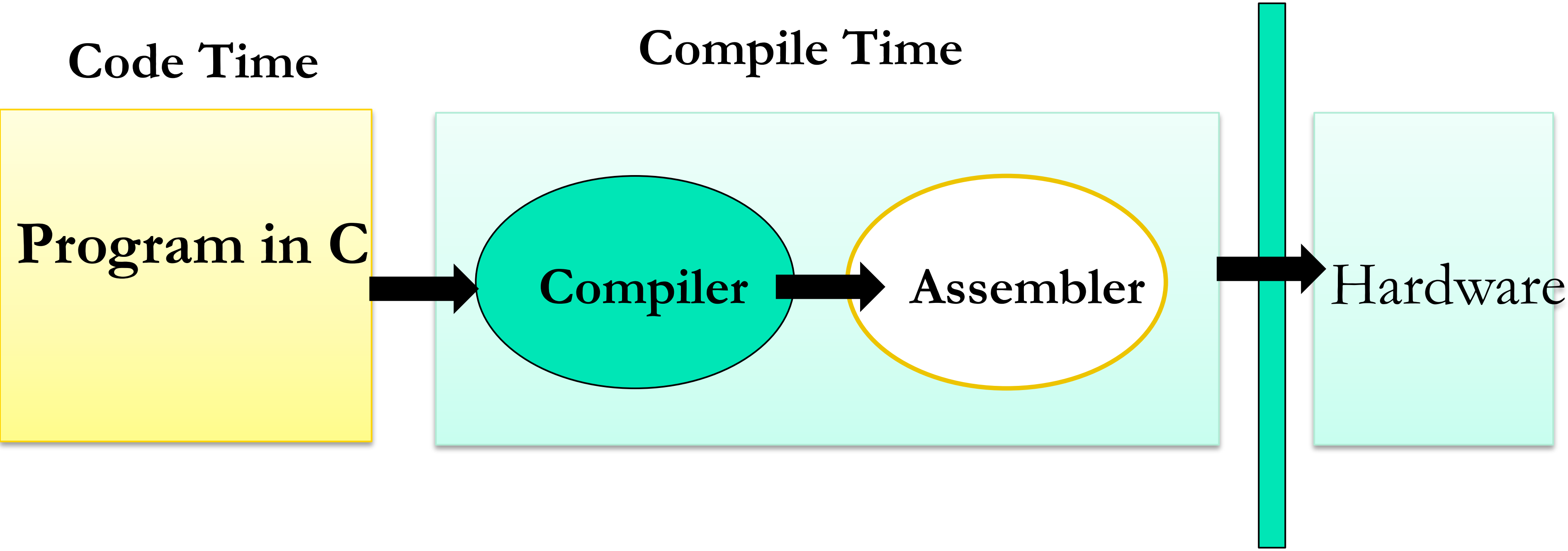
B. A symbolic representation of machine instructions

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

Steps in program translation



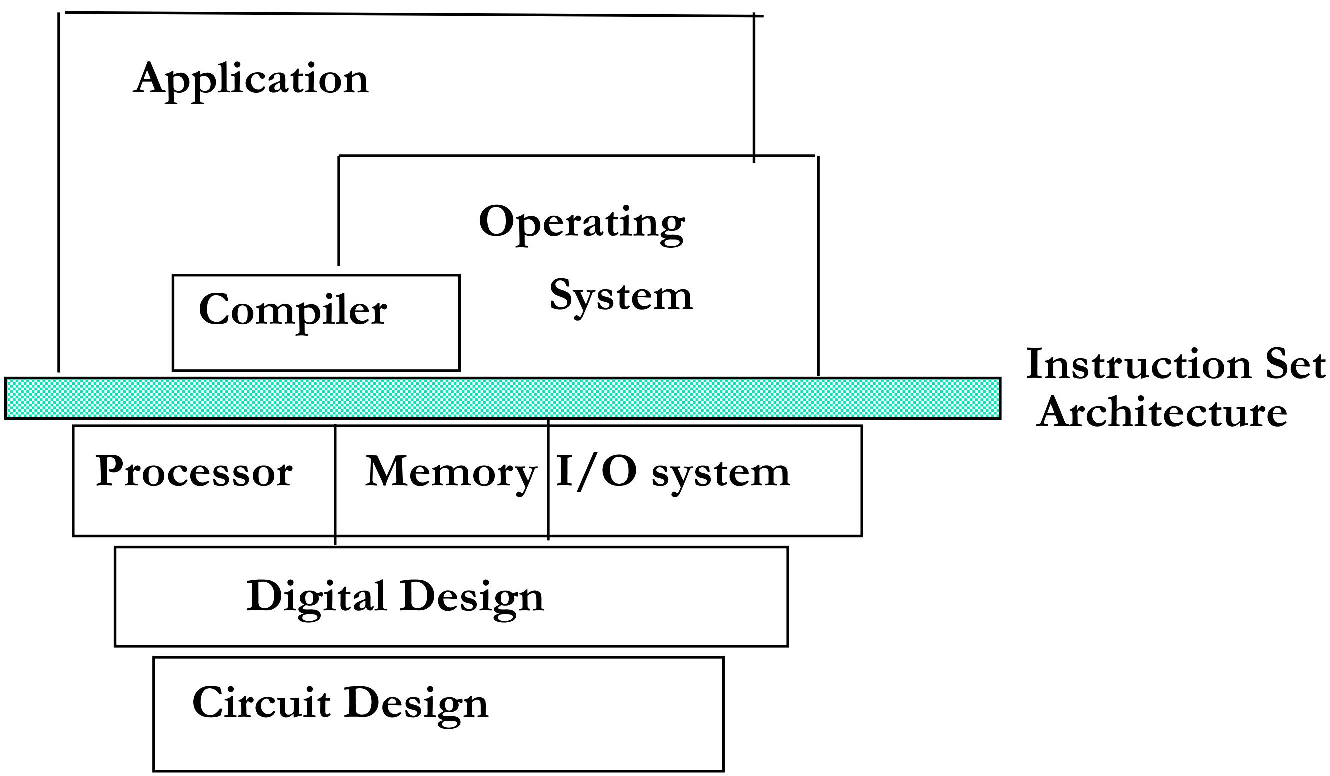
Instruction
Set
Architecture

What makes programs run fast?

1. Algorithm
2. Compiler Translation to Machine code
3. ISA and hardware implementation

What is the Instruction Set Architecture?

- Agreed-upon interface between all the software that runs on the machine and the hardware that executes it
- Primitive set of instructions a particular CPU implements

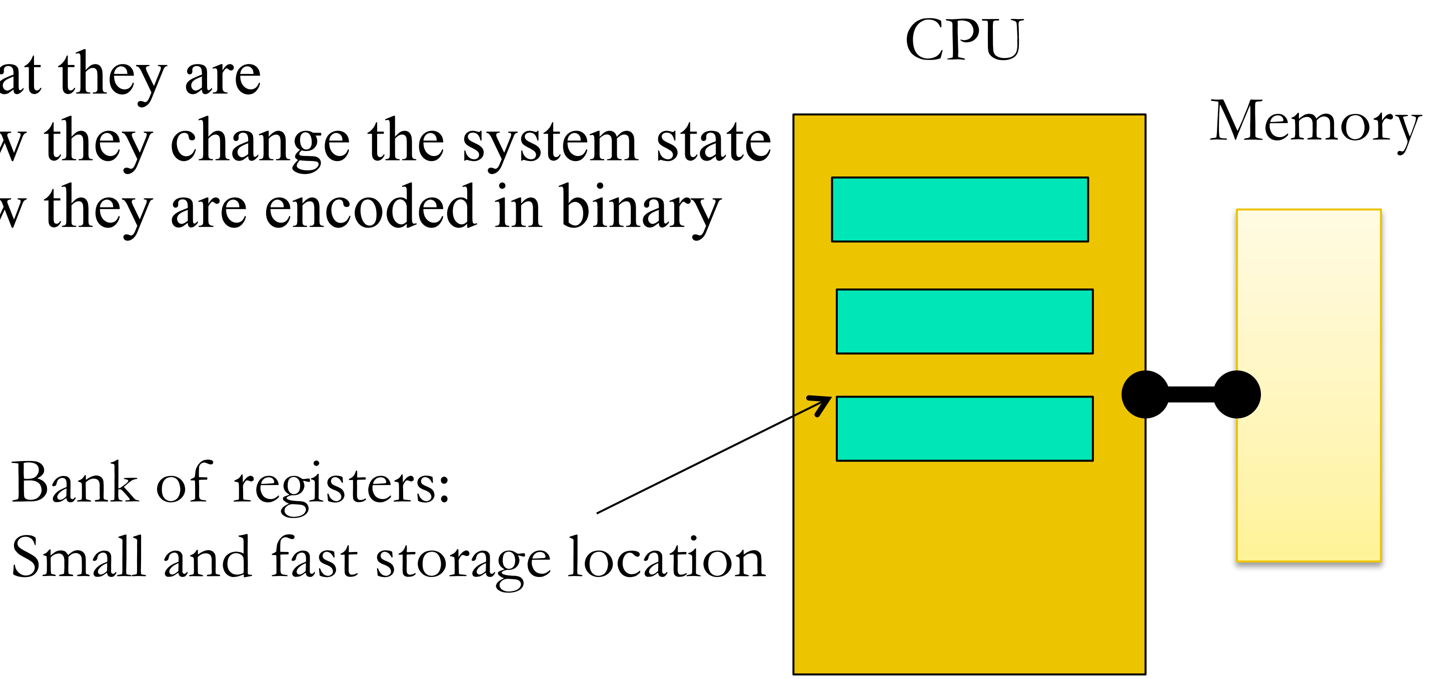


General ISA Design Aspects

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



Is the ISA different for different CPUs?

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

RISC vs CISC

RISC

- More work for compiler
- More RAM used to store instructions
- Easier to debug: more reliable processors
- Easier to optimize
  - One clock cycle per instruction

CISC

- Less work for compiler
- Fewer instructions to store
- Harder to maintain and debug

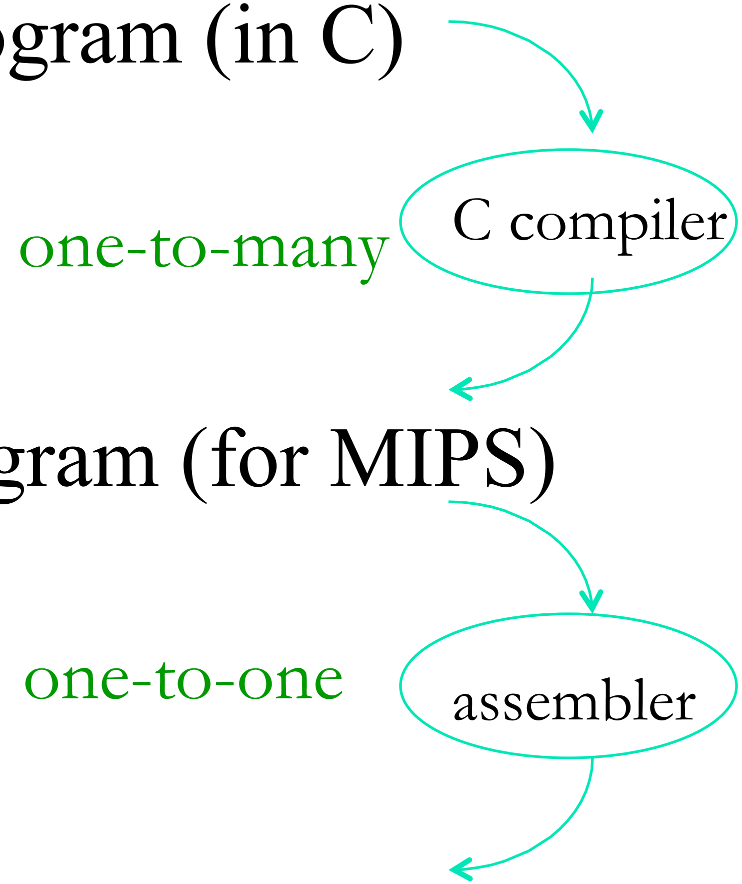
For a program written in a high-level-language, the conversion from ____ to _____ will be __________ for different target processors

	High level Language to Assembly	Assembly to Machine Language
A	One-to-Many	Many-to-Many
B	Many-to-Many	One-to-One
C	One-to-Many	One-to-One
D	One-to-One	One-to-One

Translations

- High-level language program (in 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)

```
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
 . . .
```

Architecture vs Micro-architecture

Architecture:

- Parts of processor design needed to write programs in assembly
- What is visible to s/w

  E.g Number of registers

Micro-Architecture:

- Detail of how architecture is implemented

  E.g Core frequency of the processor

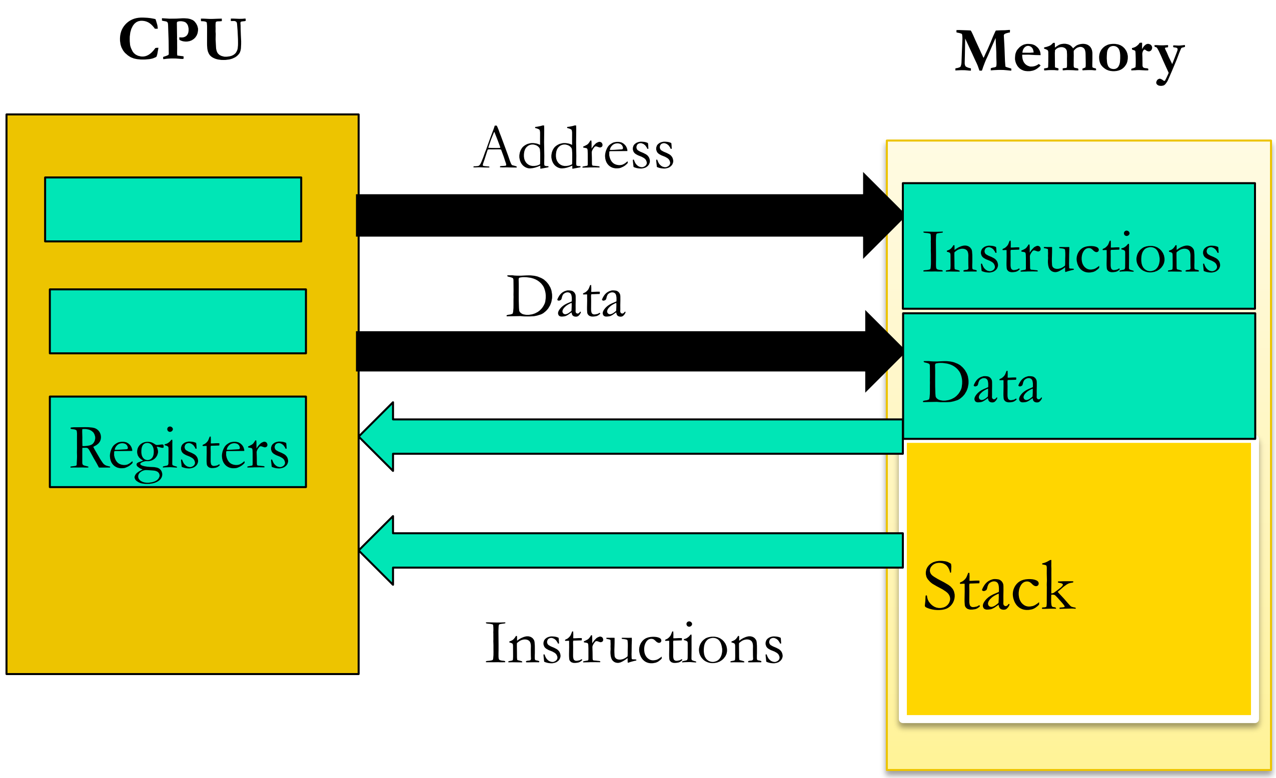
  Aside: Processor Speed:

  Intel Core i7: 1.8 GHz

  1.8 billion cycles per second

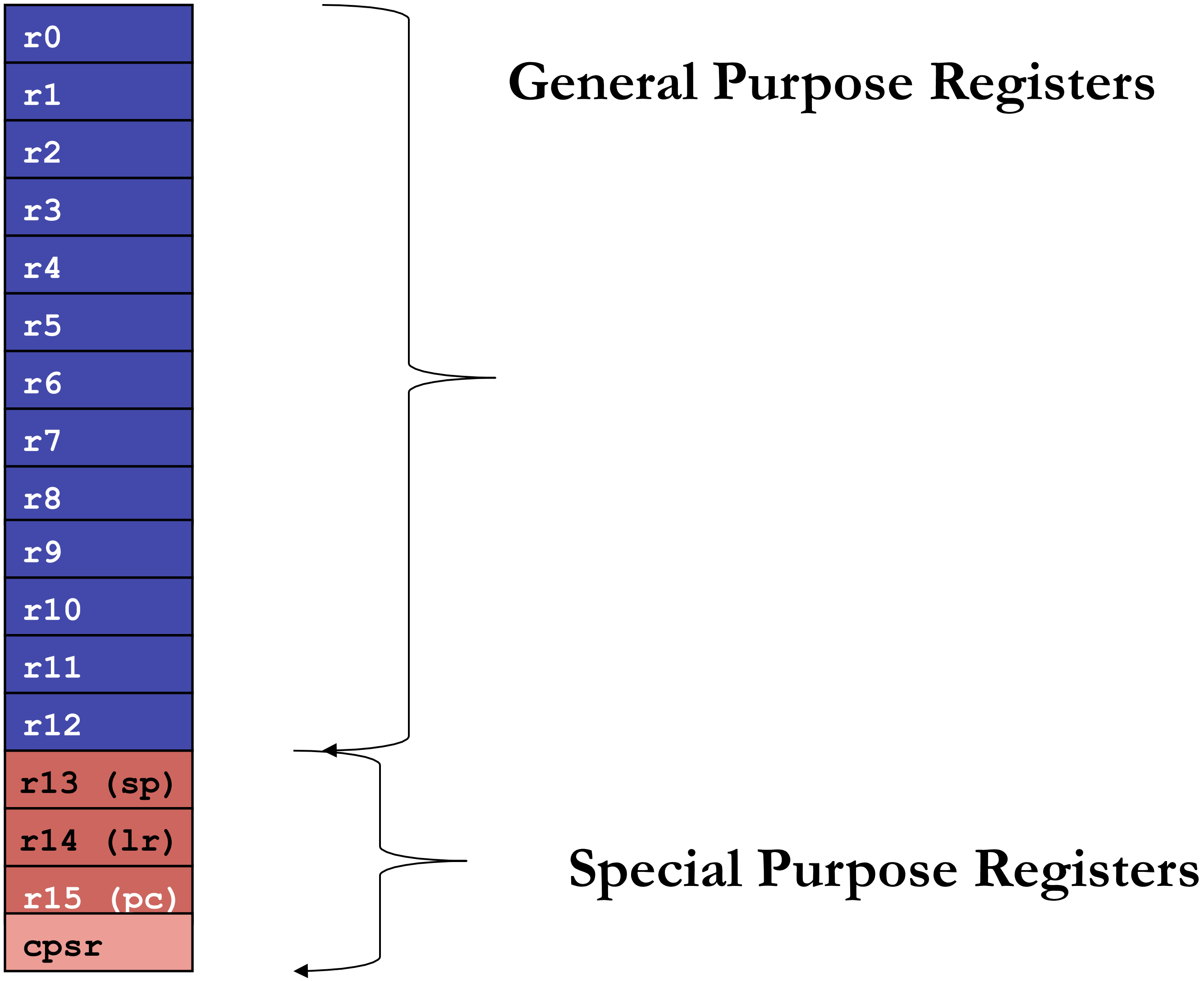
  Each instruction takes some number of cycles

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



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

C, Java Variables vs. Registers

- 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

Which one of the following is an optimization that is typically done by ARM compilers?

A. Put the address of frequently used data in registers

B. Put the value of frequently used data in registers

C. Put as much data as possible in the registers

D. Put as few program data in the registers

E. Registers are not available to the programmer/compiler – they are only used internally by the processor to fetch and decode instructions

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.

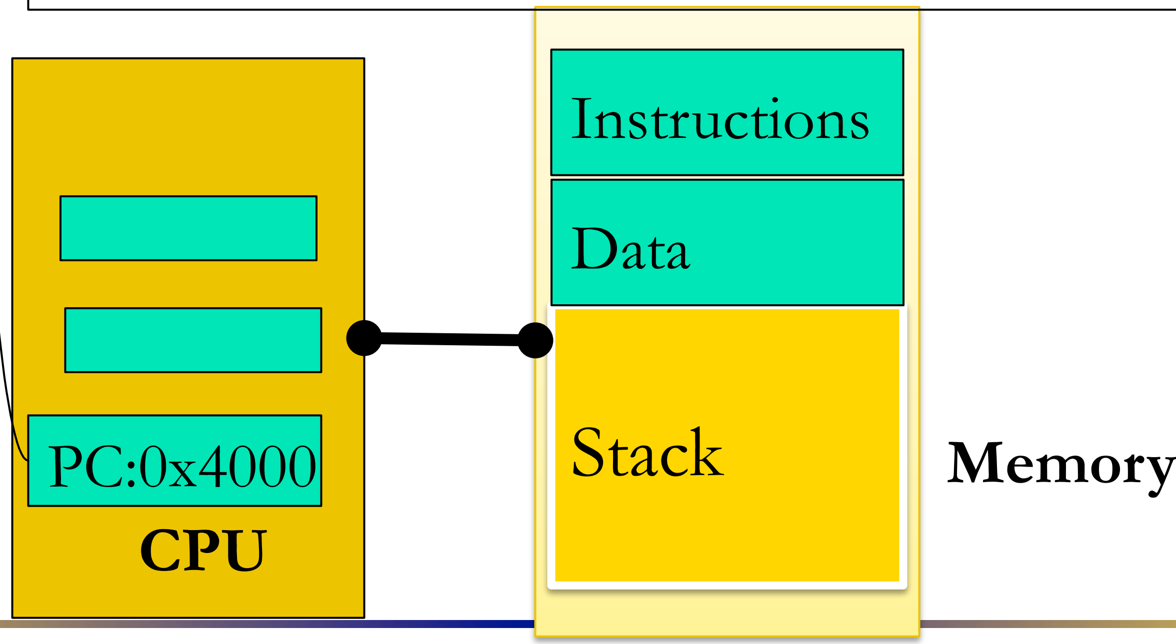
B. Are commands the computer performs.

C. Were what people originally wrote programs using.

D. All of the above.

Steps in program execution

```
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
```



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.

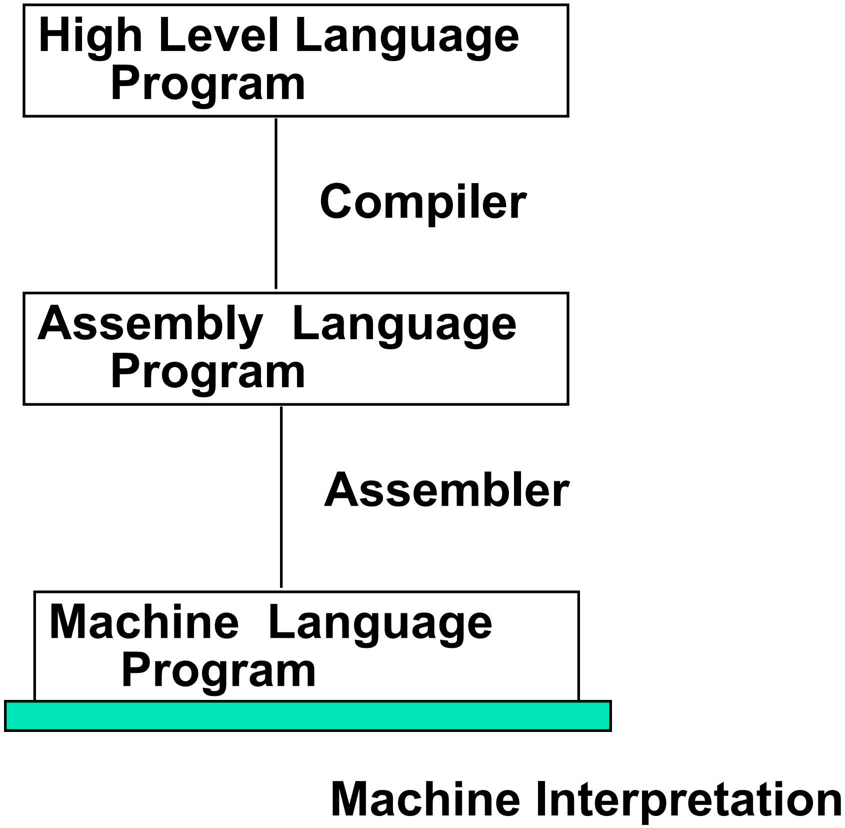
Basic Types of Instructions

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

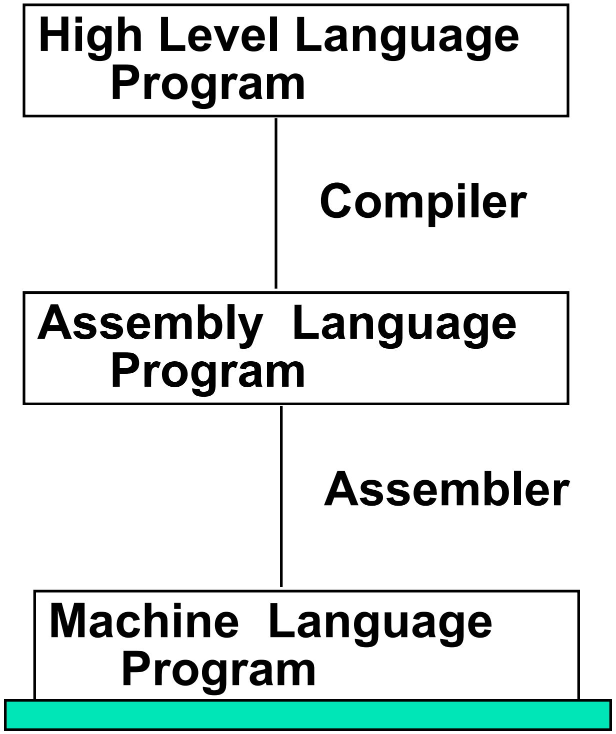
3. Control Transfer Instructions: Change flow of execution
    1. jump to another instruction
    2. conditional jump (e.g., branch if registeri == 0)
    3. jump to a subroutine

How to Speak Computer




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

How to Speak Computer

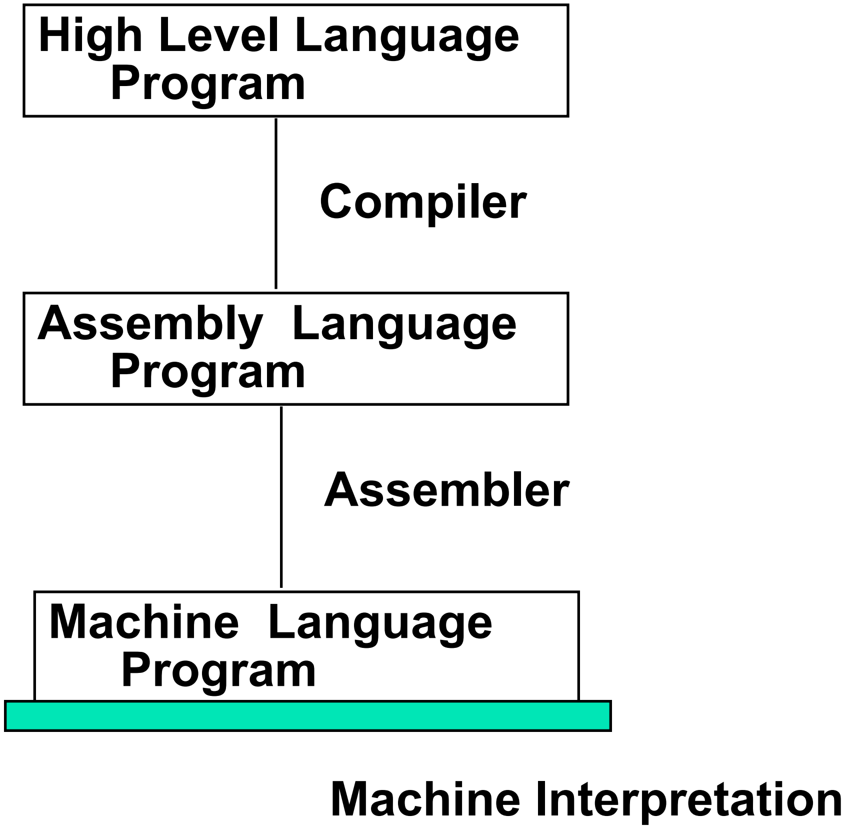


Machine Interpretation

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw  $15,   0($2)
lw  $16,   4($2)
sw  $16,   0($2)
sw  $15,   4($2)
```

How to Speak Computer




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw  $15,   0($2)
lw  $16,   4($2)
sw  $16,   0($2)
sw  $15,   4($2)
```

```
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
```

How to Speak Computer

High Level Language Program

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

Compiler

Assembly Language Program

lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)

Assembler

Machine Language Program

10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100

Machine Interpretation

Machine does something!