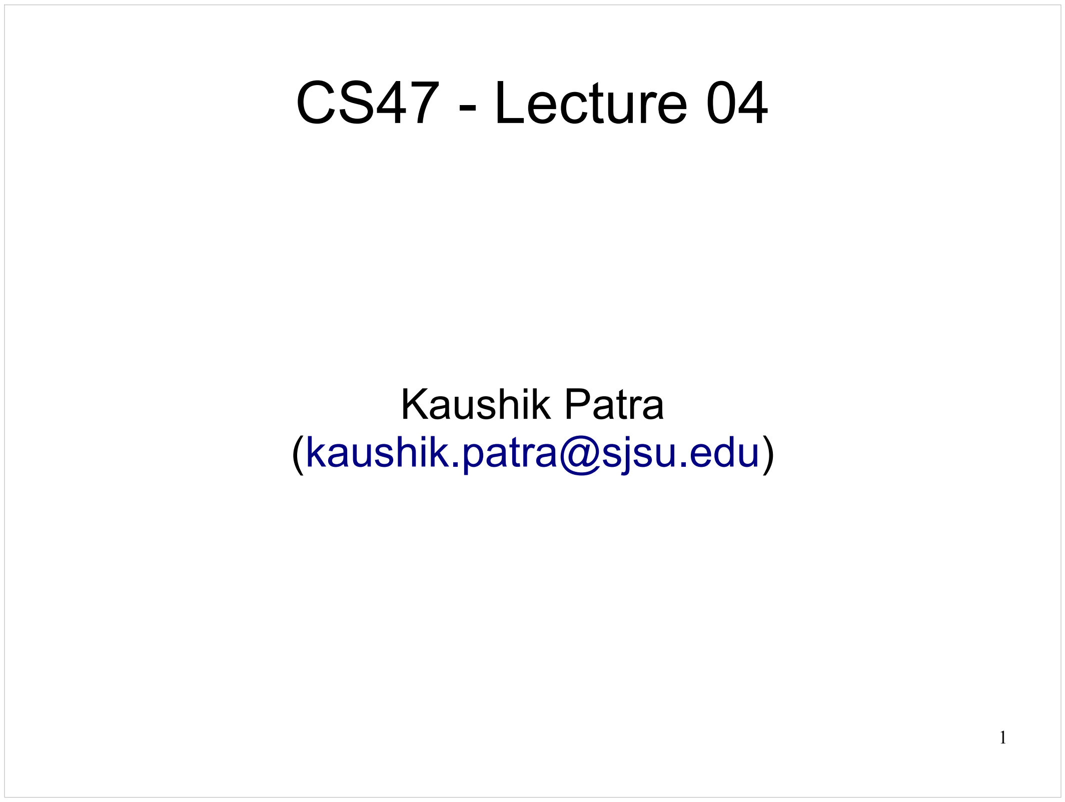


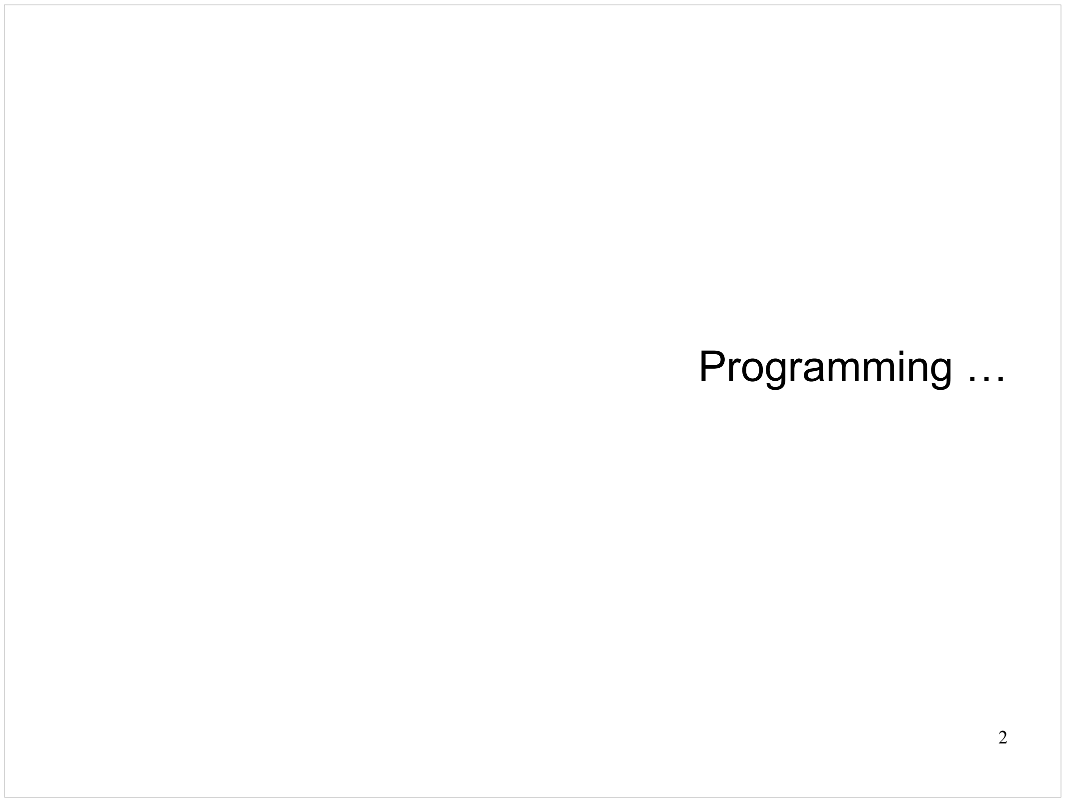
CS47 - Lecture 04

Kaushik Patra
(kaushik.patra@sjsu.edu)

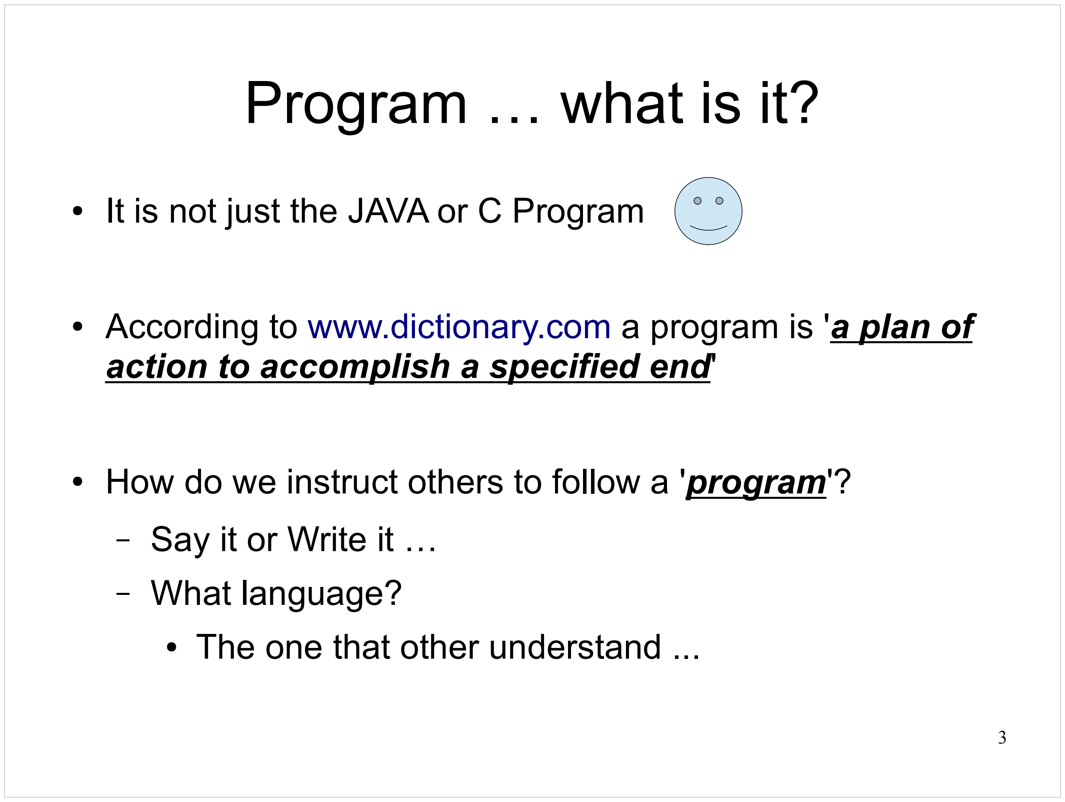
1

- Topics
  - Programming
  - Programming Language
  - Assembly Code

[Chapter A.1 of Computer Organization & Design by Patterson]



Programming …



Program … what is it?

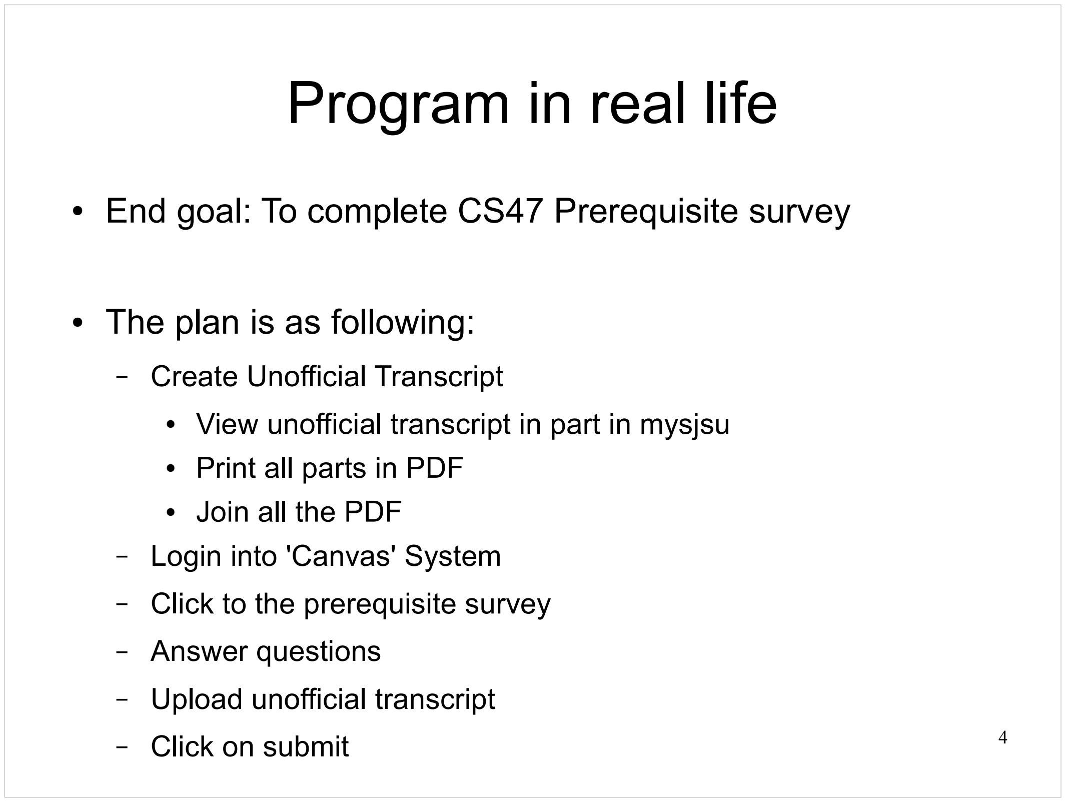
- It is not just the JAVA or C Program
- According to www.dictionary.com a program is '***a plan of action to accomplish a specified end***'
- How do we instruct others to follow a '***program***'?
  - Say it or Write it …
  - What language?
    - The one that other understand ...

3

- We all do program knowingly or unknowingly during our day to day life. For example, a student has some plan / schedule from morning till evening to attend classes, doing homework or projects, attend parties and fun, etc. This involves certain steps and flow of actions – which is nothing but a program.

- Either we do program for ourselves or others. The above example of student is programming for a student for him(her)self. Take another example of your graduate adviser. (S)He does plan for your steps to complete degree. This involves sequence of subjects and when to take those without being dropped due to incomplete prerequisite. This is also one kind of programming in real life.

- A program or plan of action should be written down (especially if it is made for others) for record – this is nothing but writing a program. A program should be written down in a language that both parties (the one who is creating and the one who is executing the plan/program) understand.



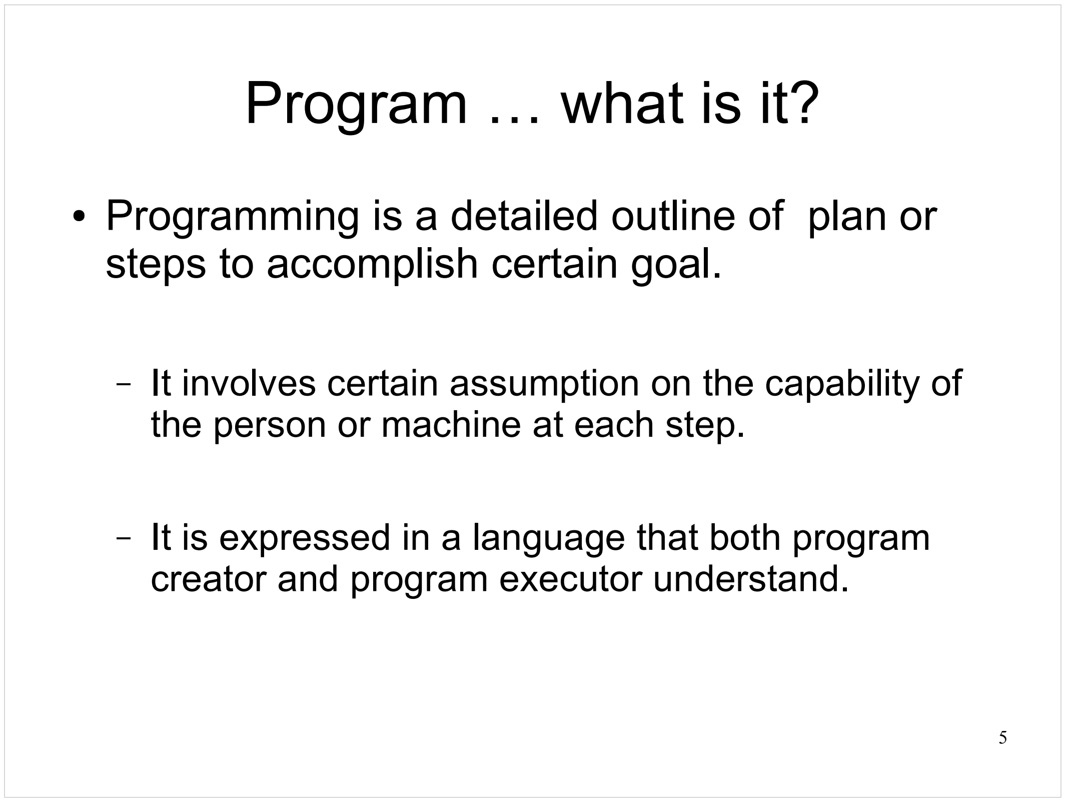
Program in real life

- End goal: To complete CS47 Prerequisite survey
- The plan is as following:
  - Create Unofficial Transcript
    - View unofficial transcript in part in mysjsu
    - Print all parts in PDF
    - Join all the PDF
  - Login into 'Canvas' System
  - Click to the prerequisite survey
  - Answer questions
  - Upload unofficial transcript
  - Click on submit

4

- If we review the steps critically, we can see that some assumptions are made at each step. The assumption is that who ever is executing the steps knows how to perform it. This is very important to understand. As a programmer, one needs to be very clear on what is the capability of the executor of the program.

- In this example, we are assuming that a student already knows how to view unofficial transcript. This may not be true for a student who is just enrolled to the SJSU system. Additionally someone may not know how to print transcript in PDF or how to join PDF files. If that is the case, then we need elaborate each such steps in sub-steps.



Program … what is it?

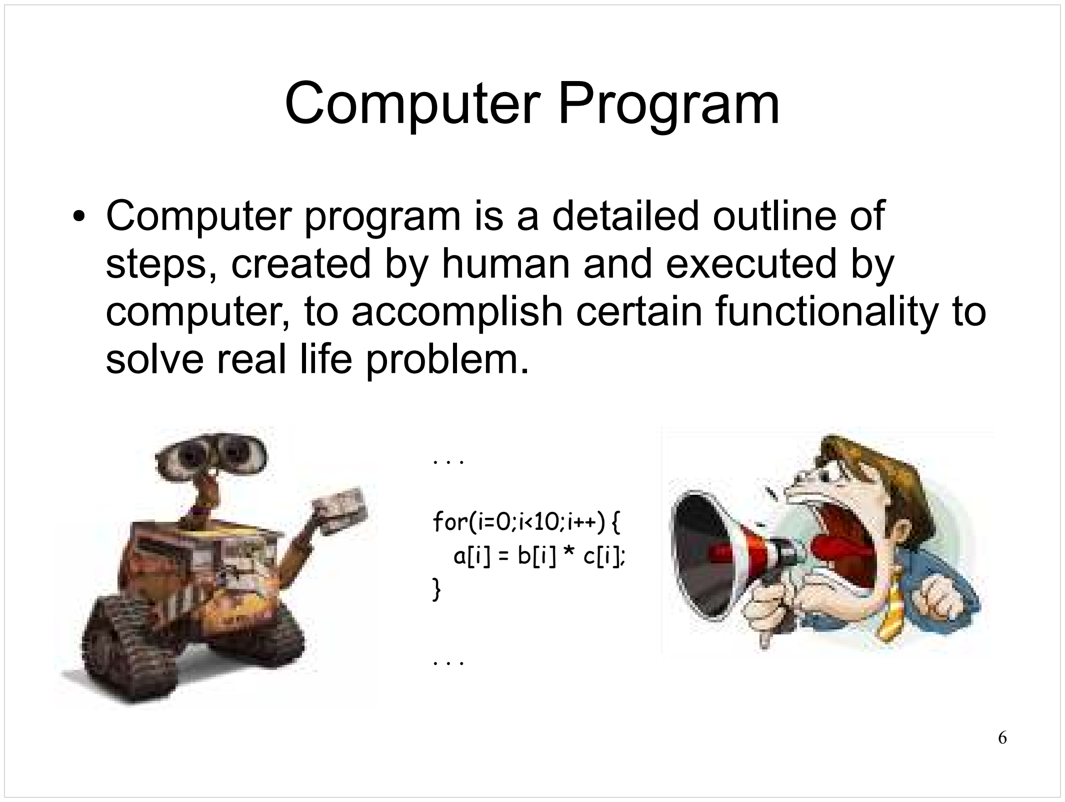
- Programming is a detailed outline of plan or steps to accomplish certain goal.
  - It involves certain assumption on the capability of the person or machine at each step.
  - It is expressed in a language that both program creator and program executor understand.

5

- A program's goal has to be very defined and measurable – sometimes, at least in real life, goal can be very abstract – like 'being happy'. If 'being happy' is a goal, it is probably can not be programmed by others and success can not be measured by others. However, if the goal is 'to make lemonade,' it is a very defined and measurable.

- The level of details depends on the program executor's capability of what it / (s)he can do. Sometime, a programmer needs to stay within boundary of limitation imposed by the executor's capability. For example, if one wants to describes steps to solve differential equation and make them executed by a 1st grader student, it will be next to impossible to create those steps. The limitation is coming from the capability of a 1st grader.

- It is also important to write program / describes steps in a language that the target executor understands. It is very important to be very clear about the target language's syntax and semantic. For example, an instruction 'Discuss the process of feeding rats with teacher' is confusing. This can either mean to discuss the process of 'feeding rats with teacher' (how much eccentric it may sound) or discuss with teacher on the process of 'feeding rat' (most probably it is the case).



Computer Program

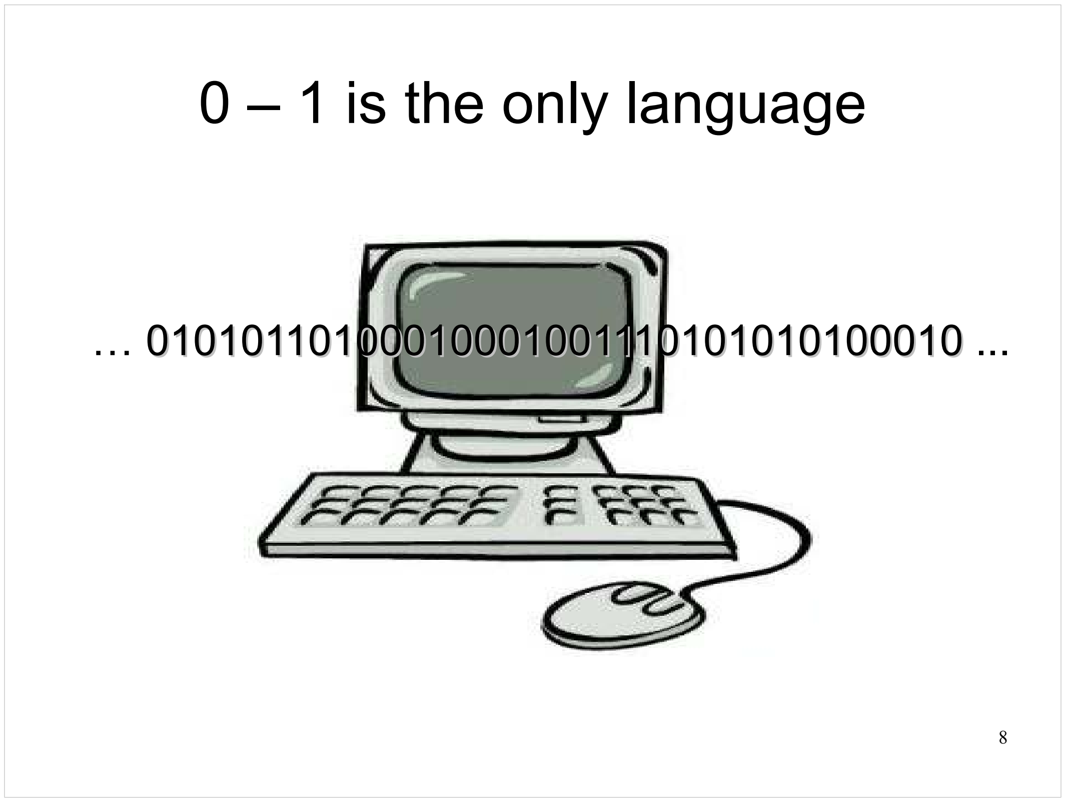
- Computer program is a detailed outline of steps, created by human and executed by computer, to accomplish certain functionality to solve real life problem.

```
...

for(i=0;i<10;i++) {
   a[i] = b[i] * c[i];
}

...
```

6

- Programming a computer is even challenging because at basic a computer processor can only do couple of hundred actions (some cases it is even less that hundred). However, these actions are very generic and can be combined into building up very complex action with different output interpretation. For example result of a sum operation (which is very basic and trivial to computer) can be either interpreted as 'prediction in stock value' or 'amount of angle to turn an automated car'. Therefore, it is programmer's responsibility to define the steps in terms of basic operation, supported by the target computer, and interpret the result to fit the end goal.

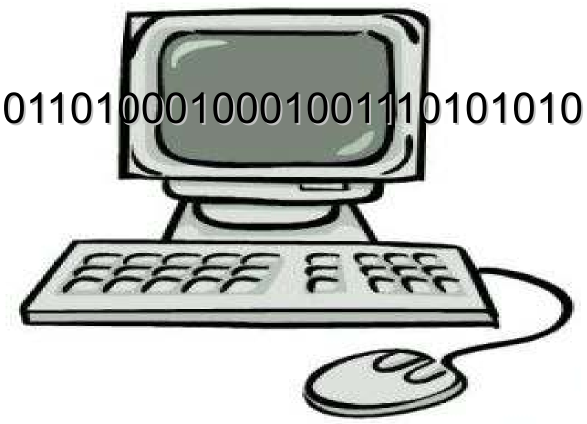
Programming Language …



0 – 1 is the only language

… 0101011010001000100111010101010100010 ...

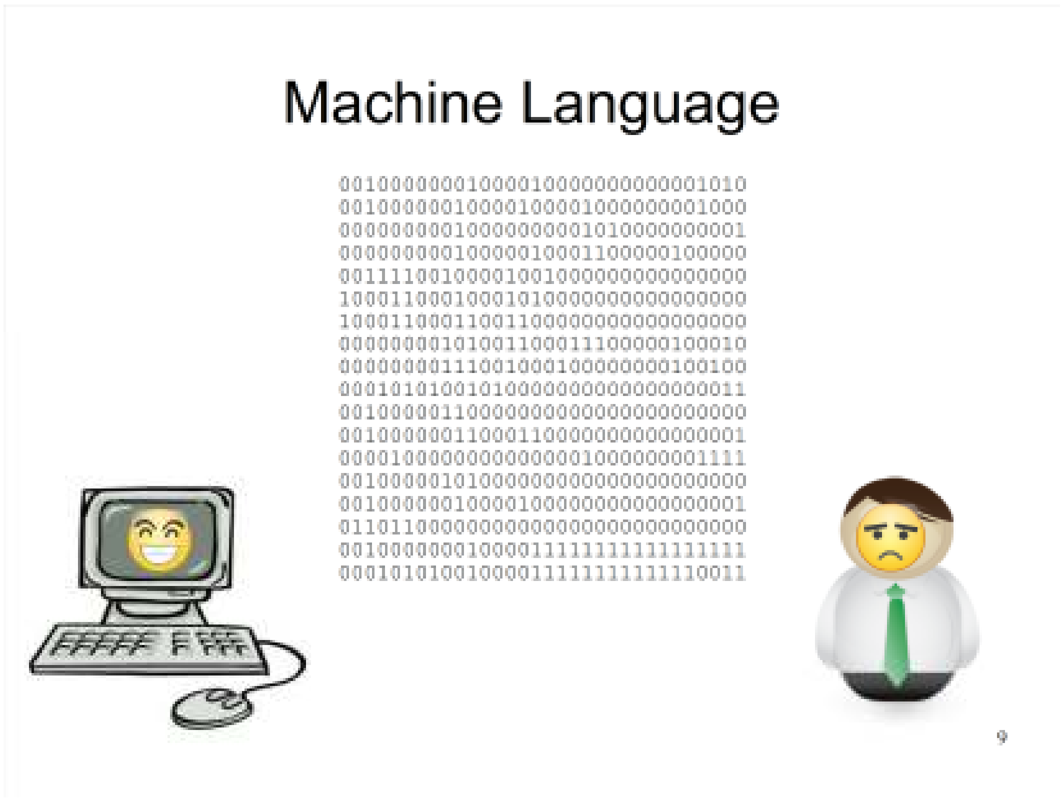
8



• At very basic level an electronic computer understands a language comprised of only two symbols – '0' and '1'. All the instructions that an electronic computer can follow and perform have to be expressed in terms of 0s and 1s.

• However, 0 and 1 is just a concept - what is the physical implication of it? In electronic world the presence of electric voltage is considered to be '1' and absence of electric voltage is considered to be '0'. Electronic circuit can be constructed to manipulate logic '0' (which is absence of voltage) and logic '1' (which is presence of voltage) into various logic and mathematical functions.

• Since Binary number system has only two symbols 0 & 1, which can correspond to the 0 and 1 of an electronic computer, binary number system and corresponding binary mathematics is the most important to study to understand computer systems and its function.



Machine Language

```
00100000001000010000000000001010
00100000010000100001000000001000
00000000010000000001010000000001
00000000010000010001100000100000
00111100100001001000000000000000
10001100010001010000000000000000
10001100011001100000000000000000
00000000101001100011100000100010
00000000111001000100000000100100
00010101001010000000000000000011
00100000110000000000000000000000
00100000011000110000000000000001
00001000000000000001000000001111
00100000101000000000000000000000
00100000010000100000000000000001
01101100000000000000000000000000
00100000001000011111111111111111
00010101001000011111111111110011
```

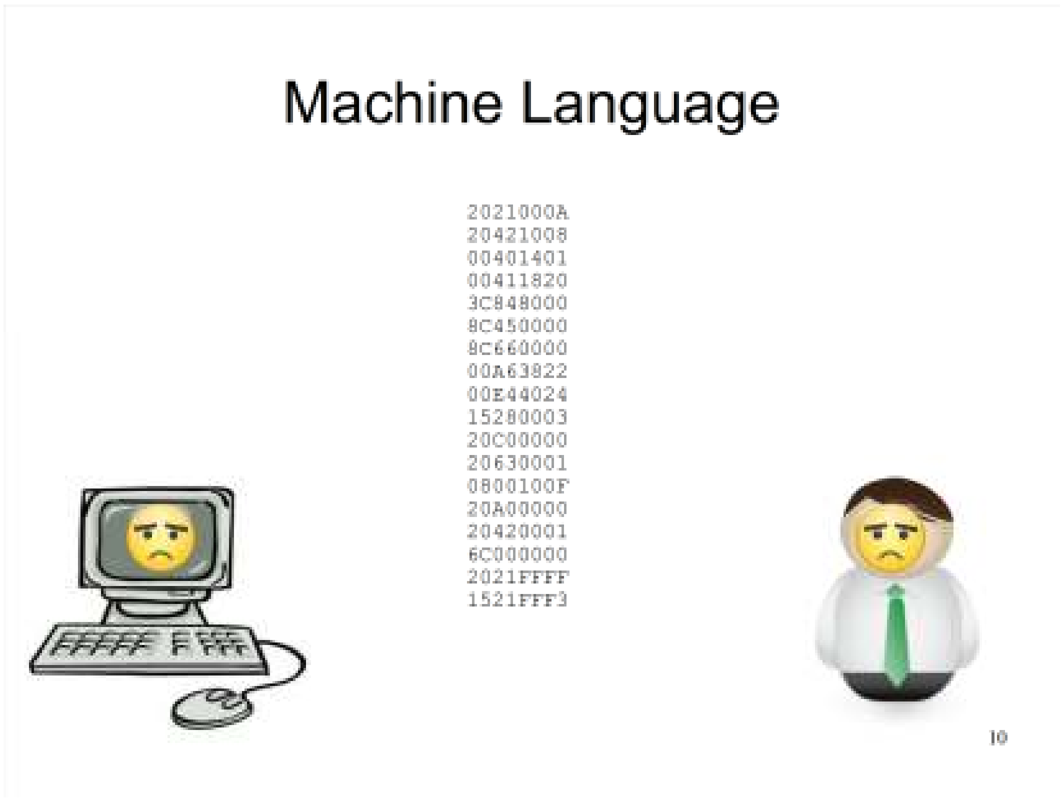
9

• Any instruction in a program has to be described in terms of 1 and 0. Each sequence pattern of 1 and 0 maps to specific instruction to computer which computer can perform and gives back result (also in terms of sequence of 1 and 0). The result sequence of 1 and 0 is interpreted by human as target application needs (e.g. same output sequence of 10110111 can be interpreted as a pixel color on display or a character equivalent to be displayed depending application)

• The sequence of 1 and 0 describing specific operation / instruction in a computer is known as machine code. This sequence of 1 and 0 of machine code can be of uniform length (which is usually same length of the word size in that computer) or of variable length.

• RISC (Reduced Instruction Set Computer) style 32-bit computer implements machine instruction / code length of 32-bits uniform across different instructions.

• CISC (Complex Instruction Set Computer) style computer (x86 machines for example) implements machine code with variable length. Length of the code depends on type of instruction.



Machine Language

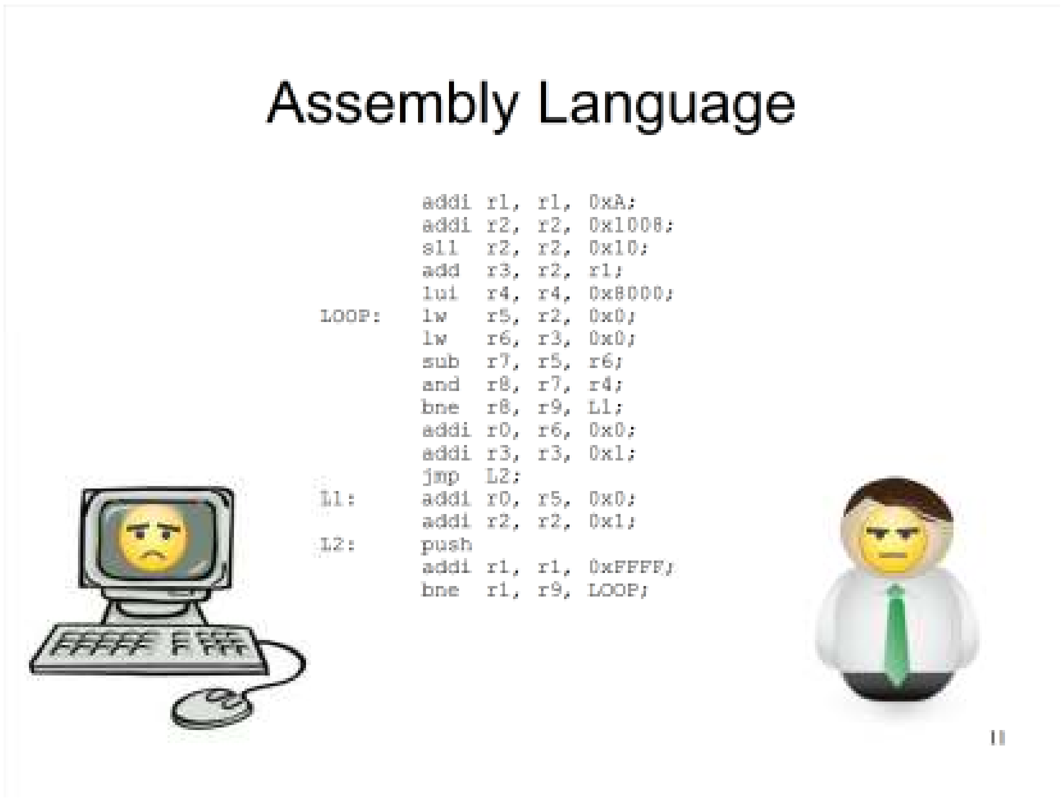
```
2021000A
20421008
00401401
00411820
3C848000
8C450000
8C660000
00A63822
00E44024
15280003
20C00000
20630001
0800100F
20A00000
20420001
6C000000
2021FFFF
1521FFF3
```

10

• The same sequence of bits (which is an word) can be expressed in terms of Hexadecimal number (assuming the target machine implements single word in terms of number of bits multiple of 4 – remember we group four bits into one hexadecimal symbol) systems. This representation of machine code is compact in nature. It takes less space to store the code as text file. Program debugger tool can also displays bit sequences in hexadecimals or binary.

• This is the reason why the study of hexadecimal number systems and related mathematical operations is important in study of computer systems.

•Though hexadecimal numbers are compact to review and store, human usually can not perceive the meaning of the machine code just by looking at the hexadecimal (or binary) instruction code. Computer can not also process the hexadecimal code directly, it has to be converted to binary before it can act on the instruction.



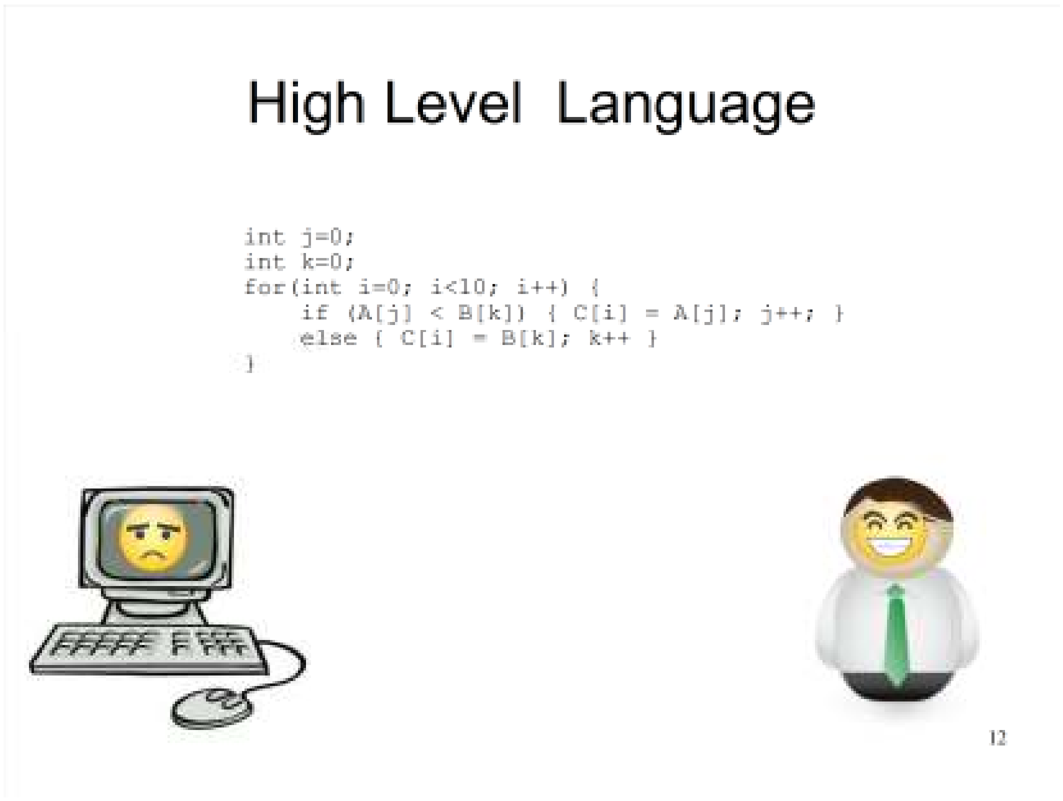
Assembly Language

```
        addi r1, r1, 0xA;
        addi r2, r2, 0x1008;
        sll  r2, r2, 0x10;
        add  r3, r2, r1;
        lui  r4, r4, 0x8000;
LOOP:   lw   r5, r2, 0x0;
        lw   r6, r3, 0x0;
        sub  r7, r5, r6;
        and  r8, r7, r4;
        bne  r8, r9, L1;
        addi r0, r6, 0x0;
        addi r3, r3, 0x1;
        jmp  L2;
L1:     addi r0, r5, 0x0;
        addi r2, r2, 0x1;
L2:     push
        addi r1, r1, 0xFFFF;
        bne  r1, r9, LOOP;
```

11

- Each machine code can be mapped directly into human readable instructions (computer can not understand this). This mapping is one to one map – the human readable form is known as assembly language. It is easier than machine code to perceive – though not as easy as perceiving a high level program (like JAVA or C).

- It is possible to write a complete program in assembly language and manually map it to corresponding machine code. In fact, in earlier days programmers need to do this to program a computer. It was tedious, time consuming job with very high chance of having bug.



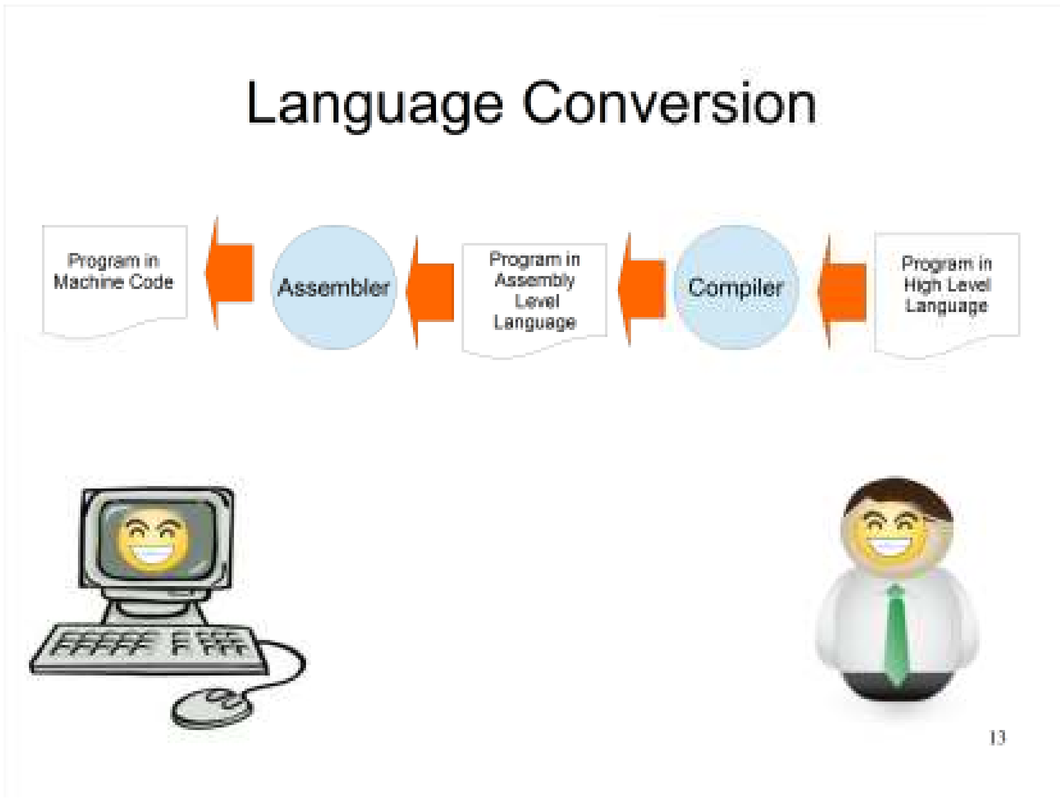
High Level Language

```
int j=0;
int k=0;
for(int i=0; i<10; i++) {
    if (A[j] < B[k]) { C[i] = A[j]; j++; }
    else { C[i] = B[k]; k++ }
}
```

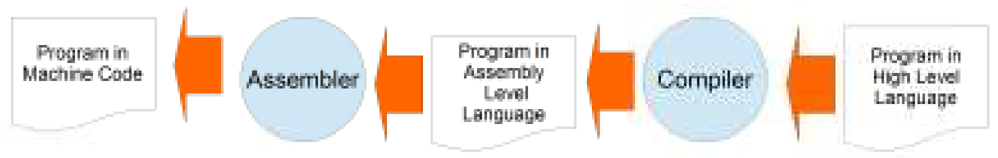
12

- Modern day programmers write program in very high level language, which are almost equivalent to natural language that we commonly can understand (e.g. English). C or JAVA is very good example of that. Unfortunately computer can not understand a bit of this high level language.

- What can we do to make both the world (human and computer) happy?
  - Think what / who do we need to make dialect with another person who only knows one language (say Bengali) which you do not know?
  - We use a person commonly known as interpreter in the above scenario. An interpreter is a person who can understand and talk two different languages (e.g. Bengali and English) and act as translator of one person's dialogue to other person and vice-verse.



Language Conversion


Program in Machine Code
Assembler
Program in Assembly Level Language
Compiler
Program in High Level Language

13

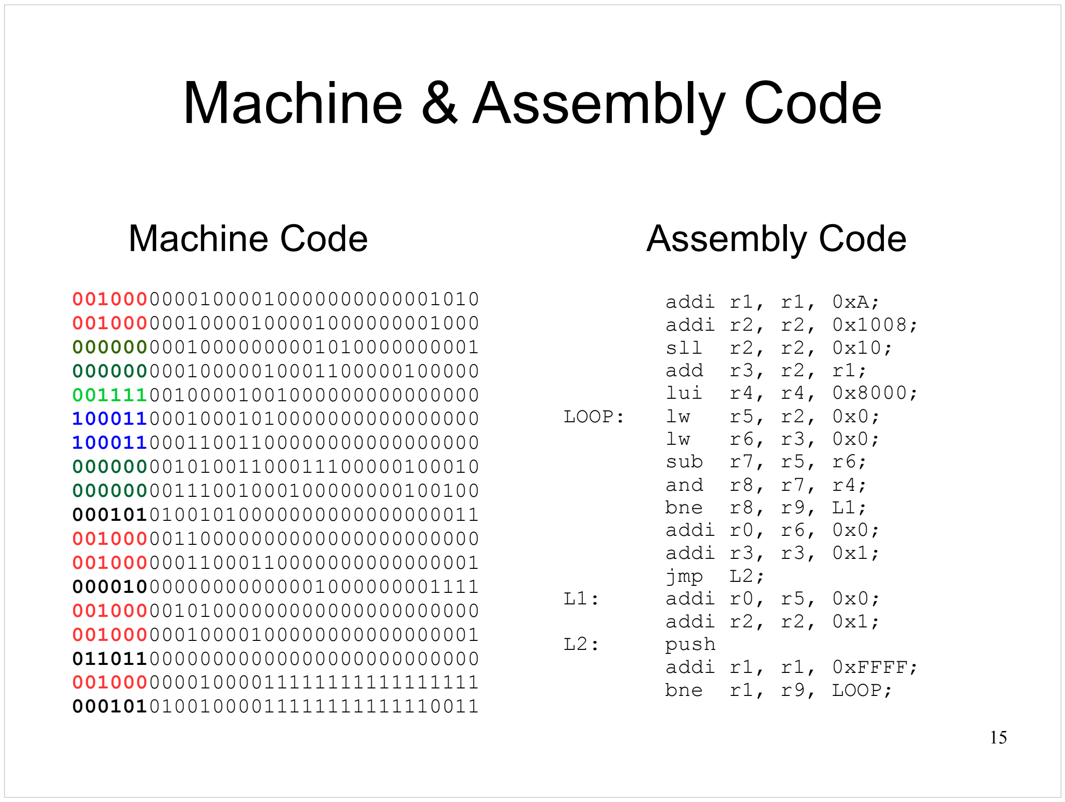
• Similar to real world, we use interpreter (or a set of interpreters) to convert program written in high level language into machine codes. There are two interpreters involved in this process of language conversion (do not mix this 'interpreter' with the 'interpreter' that people common uses in computer science to refer to PERL or TCL scripting language – we are talking here in terms of a person who acts as interpreter) - compiler and assembler.

• A compiler is an automated software that compiles a program in high level language to equivalent program in Assembly level language. Maintaining is equivalency is very important – otherwise the target application will not function as intended.

• An assembler is an automated program that maps assembly code into corresponding machine code. It is easier task compare to compiling a program in high level language.

• All modern days compiler is a combination of compilation and assembly, therefore programmer does not need to call the assembler explicitly. Thus the process of generating machine codes from a high level language is usually called 'compilation' of program, though it involves both compiling and assembling.

Assembly Code …



Machine & Assembly Code

Machine Code

```
00100000001000010000000000001010
00100000010000100001000000001000
00000000010000000001010000000001
00000000010000010001100000100000
00111100100001001000000000000000
10001100010001010000000000000000
10001100011001100000000000000000
00000000101001100011100000100010
00000000111001000100000000100100
00010101001010000000000000000011
00100000110000000000000000000000
00100000011000110000000000000001
00001000000000000000100000001111
00100000101000000000000000000000
00100000010000100000000000000001
01101100000000000000000000000000
00100000001000011111111111111111
00010101001000011111111111110011
```

Assembly Code

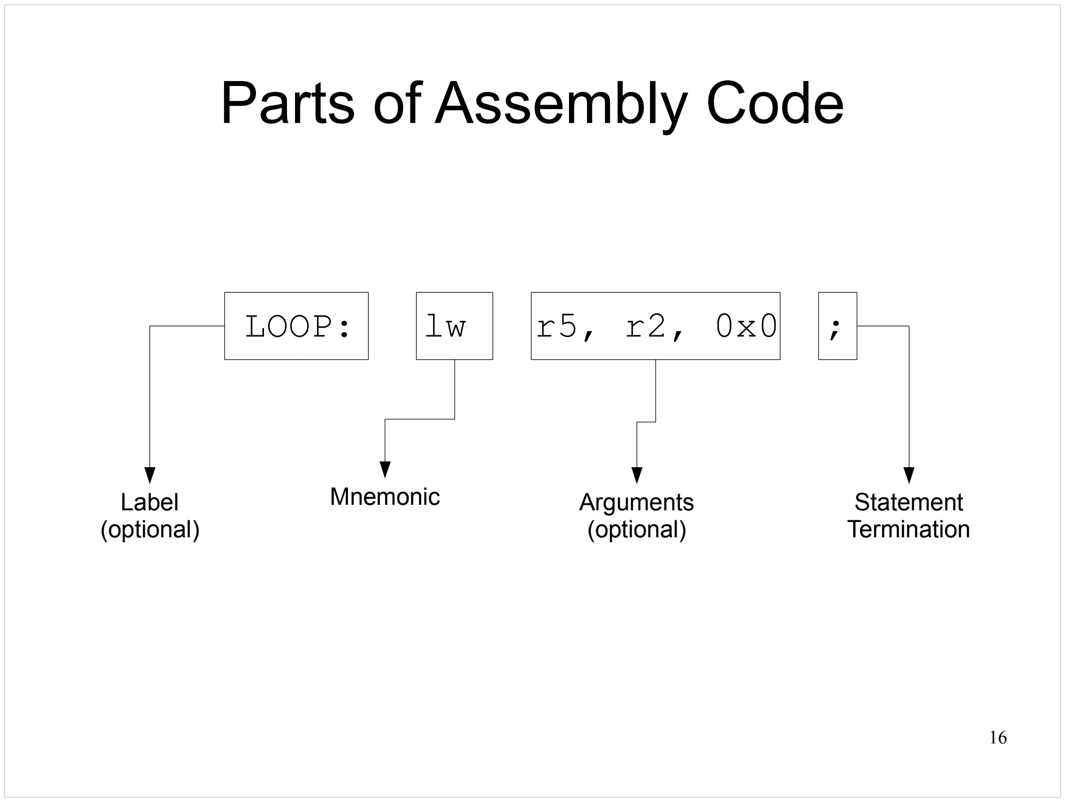
```
        addi r1, r1, 0xA;
        addi r2, r2, 0x1008;
        sll  r2, r2, 0x10;
        add  r3, r2, r1;
        lui  r4, r4, 0x8000;
LOOP:   lw   r5, r2, 0x0;
        lw   r6, r3, 0x0;
        sub  r7, r5, r6;
        and  r8, r7, r4;
        bne  r8, r9, L1;
        addi r0, r6, 0x0;
        addi r3, r3, 0x1;
        jmp  L2;
L1:     addi r0, r5, 0x0;
        addi r2, r2, 0x1;
L2:     push
        addi r1, r1, 0xFFFF;
        bne  r1, r9, LOOP;
```

15

• As indicated earlier, each machine code corresponds to an unique assembly code and vice-verse.

• In this class, we'll take MIPS (Microprocessor without Interlocked Pipe lined Stages) assembly language as reference. MIPS is a RISC style 32 bit processor implementing machine code with uniform length of 32 for the bit sequence representing the machine code.

• If we carefully observe the bit pattern of first 6 bits in the program we can see repetitive patterns. These 6 bits determines what operation to performs. This means MIPS can supports maximum of 64 operations. This set has been extended by using another 6 bit fields in the machine code for certain set of operations (we'll review it later).



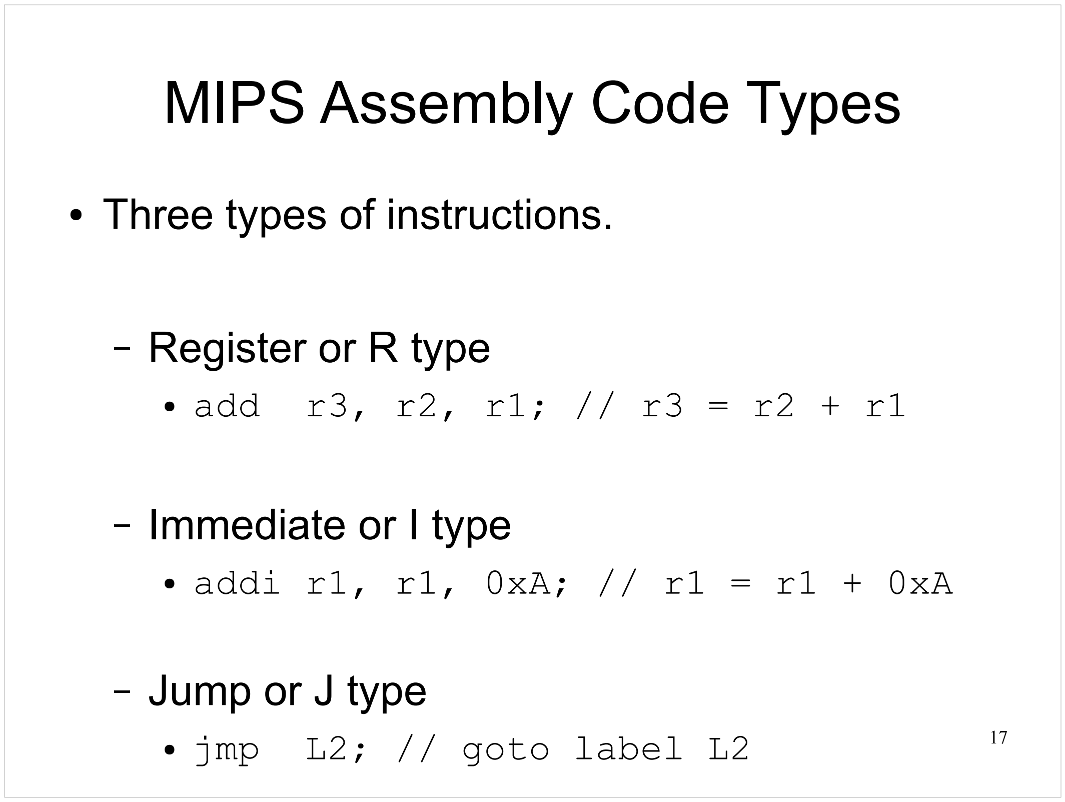
Parts of Assembly Code

• Label is not a part of machine code – it is to mark a statement with some name to refer it later (mostly used for jumping and memory reference). This label is used by assembler to determine address of that piece of code or memory storage.

• Mnemonic is the pattern of letter corresponding to the machine instruction encoded in 6-bit binary pattern. This is used to remember and identify the operation effectively. For example remembering 'lw' for load word is much easier than remembering a number 0x23.

• A mnemonic is followed by optional arguments, which can be register number, immediate information, label, or none. Each instruction has its own list of arguments.

• Each assembly statement is ended with ';' - this is considered as statement termination.

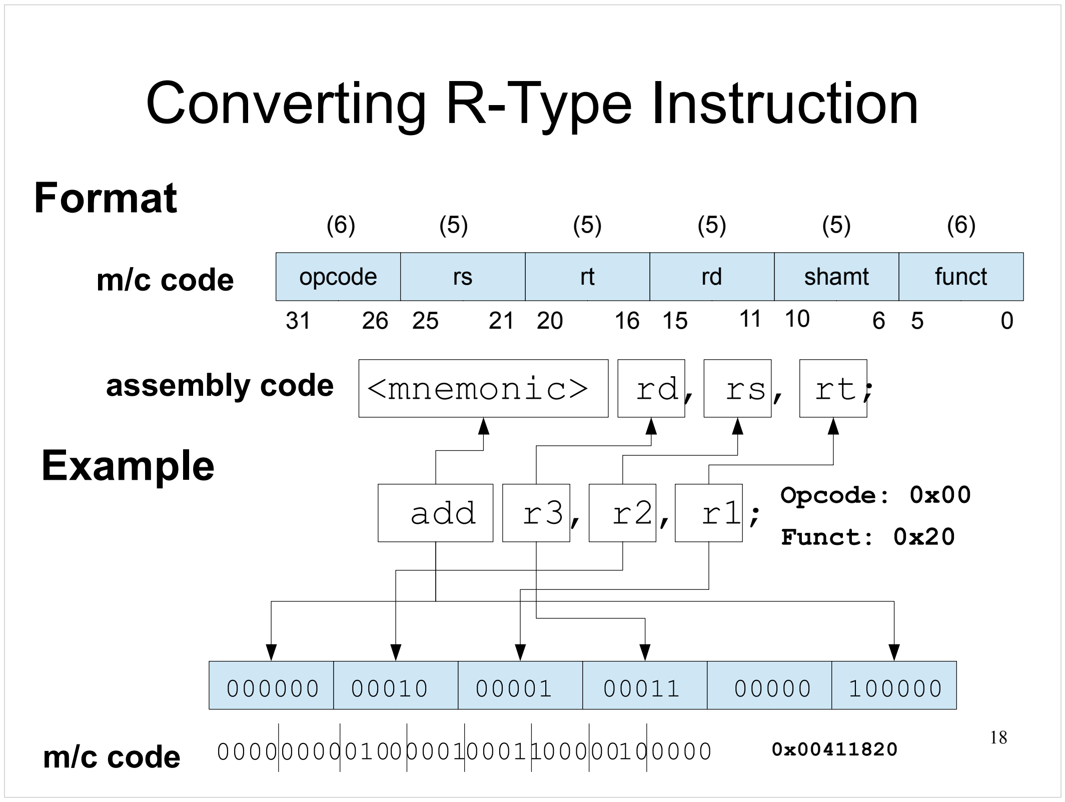


MIPS Assembly Code Types

- Three types of instructions.
  - Register or R type
    - `add  r3, r2, r1; // r3 = r2 + r1`
  - Immediate or I type
    - `addi r1, r1, 0xA; // r1 = r1 + 0xA`
  - Jump or J type
    - `jmp  L2; // goto label L2`

17

- There are three native instruction types in MIPS processor.
  - Register (R) type instruction involves operation between values stored in registers and result is also stored back into register.
  - Immediate (I) type instruction also involves registers, but one operand value is embedded within the machine code.
  - Jump (J) type instructions usually involves with Label or code address to jump to specific code location.

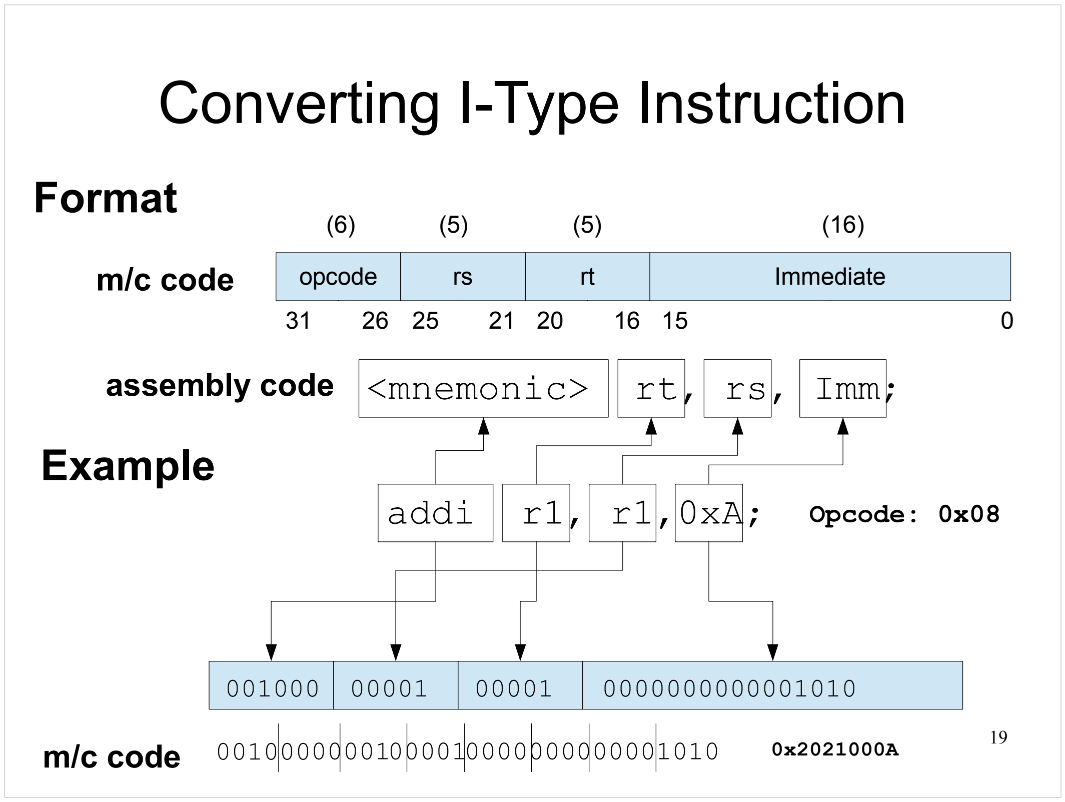


• All the R-type instructions has predefined format for the bit patterns as following.

- Bit 31-26 (6 bits) is the operation code (opcode in short). Most of the R-type instruction has opcode on 0x00. Different functionality is specified using the 'funct' field.
- Bit 25-21 (5 bits) is the register number of one of the operands (rs).
- Bit 20-16 (5 bits) is the register number of other operand (rt).
- Bit 15-11 (5 bits) is the register number of the register where the result will be stored (rd).
- Bit 10-6 (5 bits) is the shift amount embedded within instruction. Shift operation uses this field to encode the instruction.
- Bit 6-0 (6 bits) is to encode the desired function to be done (like addition, subtraction, etc.)

• To convert assembly code into machine code, we need to place the bit patterns in the field as described above corresponding to the assembly code. The hex conversion can be made by grouping 4 bits.

- Note that the sequence of the rs, rt, rd is not same as in the assembly (where sequence is rd, rs, rt) equivalent.



Converting I-Type Instruction

Format

m/c code

opcode (6)	rs (5)	rt (5)	Immediate (16)
31–26	25–21	20–16	15–0

assembly code `<mnemonic> rt, rs, Imm;`

Example

`addi r1, r1, 0xA;` **Opcode: 0x08**

```
001000  00001  00001  0000000000001010
```

m/c code `0010 0000 0010 0001 0000 0000 0000 1010` **0x2021000A**

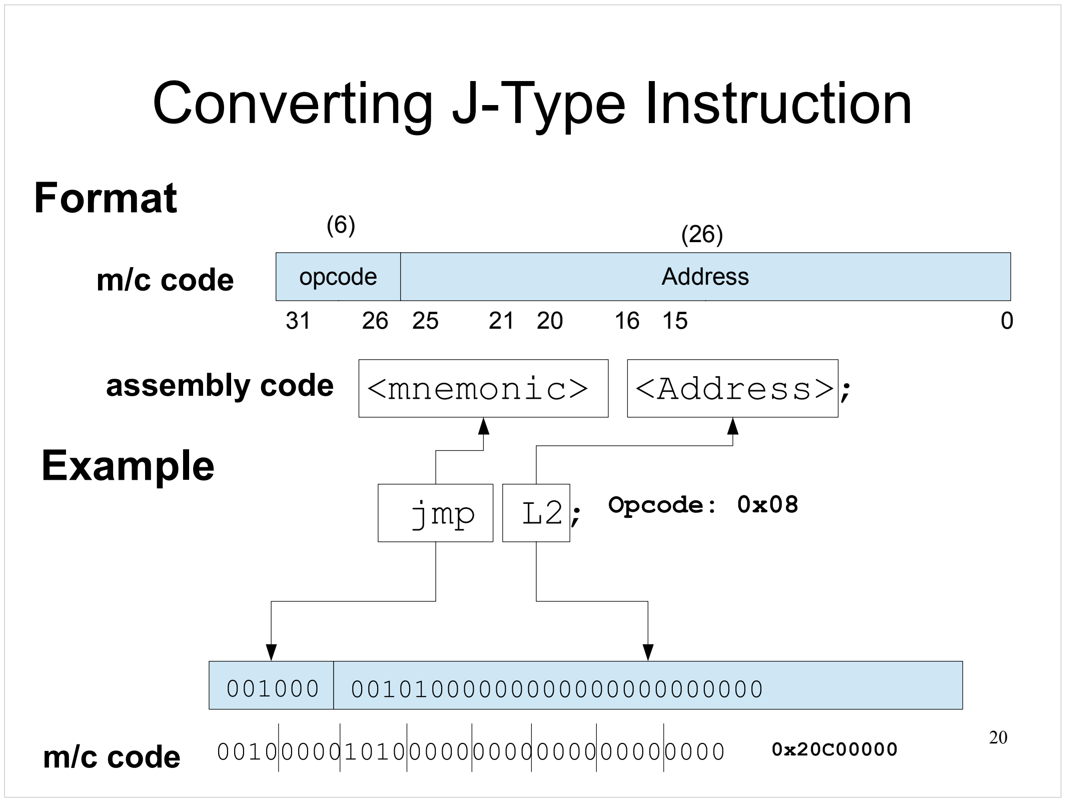
19

• All the I-type instructions has predefined format for the bit patterns as following.

- Bit 31-26 (6 bits) is the operation code (opcode in short).
- Bit 25-21 (5 bits) is the register number of one of the operands (rs).
- Bit 20-16 (5 bits) is the register number of the register where the result will be stored (rt).
- Bit 15-10 (16 bits) is the immediate value used as another operand. Since the operations are 32-bits, this 16 bit field is extended (either sign extension or zero extension) to 32-bit before the operation is done.

- To convert assembly code into machine code, we need to place the bit patterns in the field as described above corresponding to the assembly code. The hex conversion can be made by grouping 4 bits.

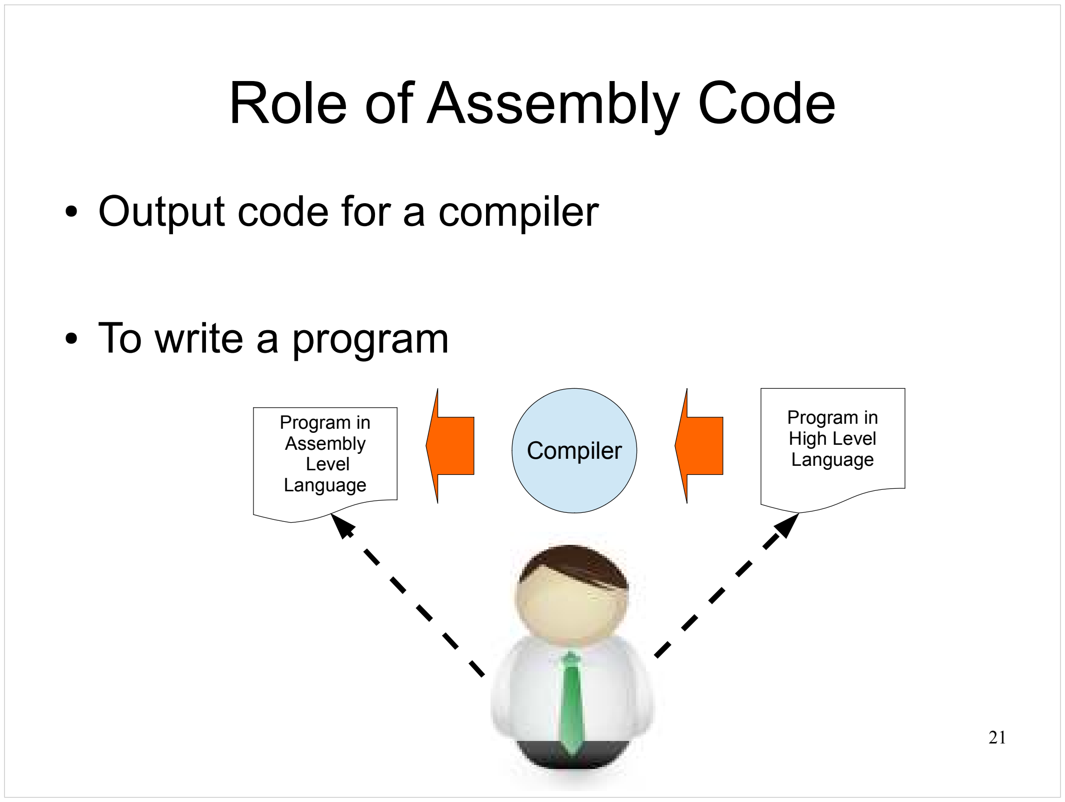
  - Note that the sequence of the rs, rt is not same as in the assembly (where sequence is rt, rs) equivalent.



• All the J-type instructions has predefined format for the bit patterns as following.

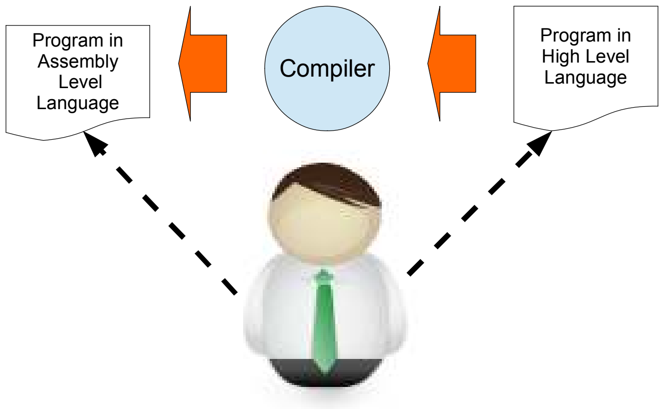
- Bit 31-26 (6 bits) is the operation code (opcode in short).
- Bit 25-0 (26 bits) is the address specifier field.

- To covert assembly code into machine code, we need to place the bit patterns in the field as described above corresponding to the assembly code. The hex conversion can be made by grouping 4 bits.



Role of Assembly Code

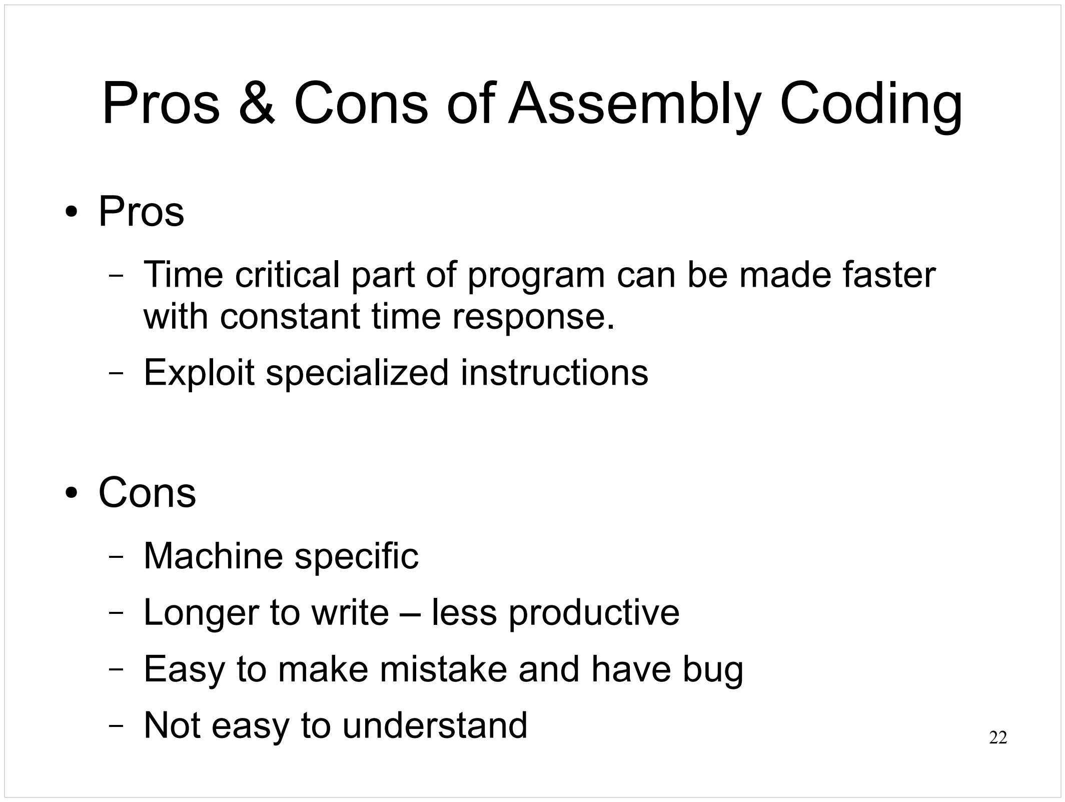
- Output code for a compiler
- To write a program


Program in Assembly Level Language
Compiler
Program in High Level Language

21

• Assembly code is output from a compiler. Compiler has two parts, front end and back end. The front end part analyzes the target program written in high level language and convert into an equivalent internal representation independent of target hardware. AST or abstract syntax tree is mostly used for this purpose. The back end part of compiler is hardware dependent and depending on the target platform, it converts the internal representation of a high level program into corresponding assembly level code.

• In certain scenario a programmer may need to code directly in assembly. Especially if the execution time is very critical in some real time application (like automated brake in a car's safety system) and the performance needs to be a predetermined fixed execution time, the assembly code can give guarantee to that. Since compiler is generic it is not guaranteed to have fixed execution time from compiler to compiler (even between different revision). Even if a non-real time application, some critical parts of the software is written in assembly to boost performance of the software.



Pros & Cons of Assembly Coding

- Pros
  - Time critical part of program can be made faster with constant time response.
  - Exploit specialized instructions
- Cons
  - Machine specific
  - Longer to write – less productive
  - Easy to make mistake and have bug
  - Not easy to understand

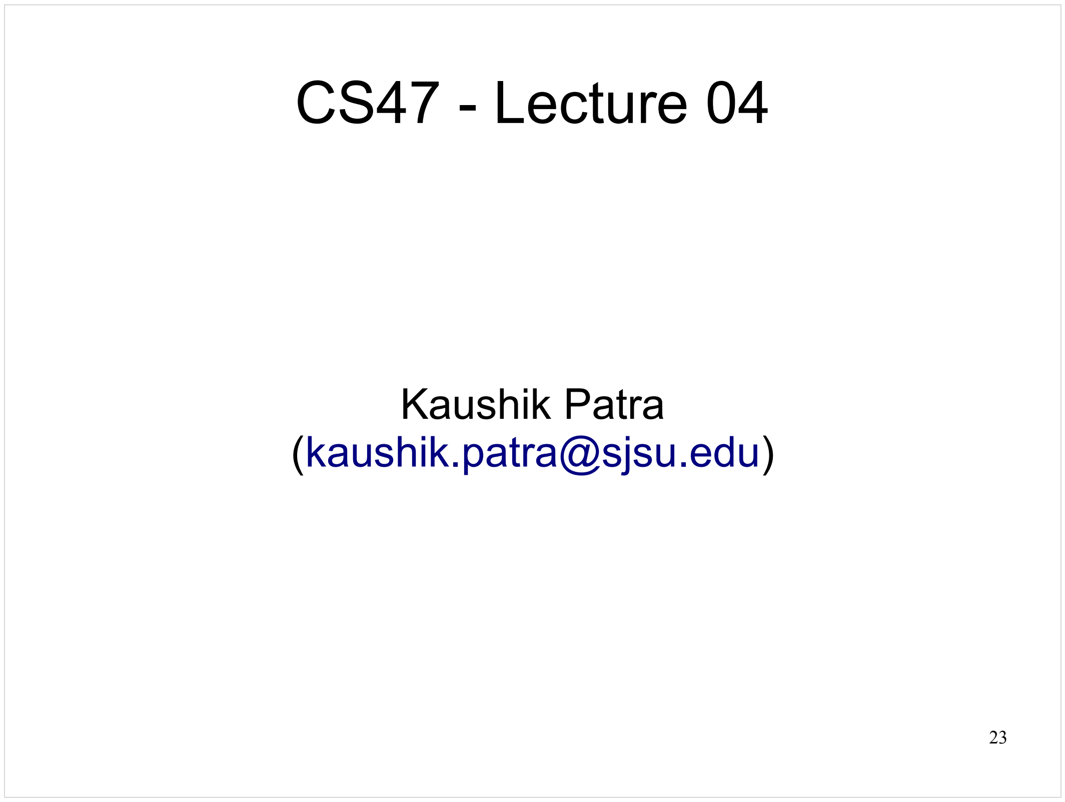
22

• Assembly code can make a part of code faster. Even it can ensure a constant time turn around. Sometime programmer can exploit very specialized assembly code (example: string copy is nothing but memory copy) which a generic compiler can not. In that respect, assembly language provides a very powerful tool to programmer's hand.

• All the assembly languages are machine specific unlike high level language like C/C++/JAVA. A high level program can be compiled into multiple target platform, but porting assembly program is very hard, if not impossible.

• The assembly program tends to be longer (at least 3 times) because assembly language provide the raw capability of a machine which is very limited. Since high level language usually wraps around / translate to group of multiple assembly code, coding in high level language is usually compact.

• Being down to almost at machine level, human's natural thinking does not go very well with assembly code – therefore it is easier to make mistakes and introduce bugs. It is hard to debug as well for being too big to handle and too cryptic to perceive. It is also not easy to understand an existing program.



CS47 - Lecture 04

Kaushik Patra
(kaushik.patra@sjsu.edu)

23

- Topics
  - Programming
  - Programming Language
  - Assembly Code

[Chapter A.1 of Computer Organization & Design by Patterson]