

5. Assembly Language

Assignment Project Exam Help

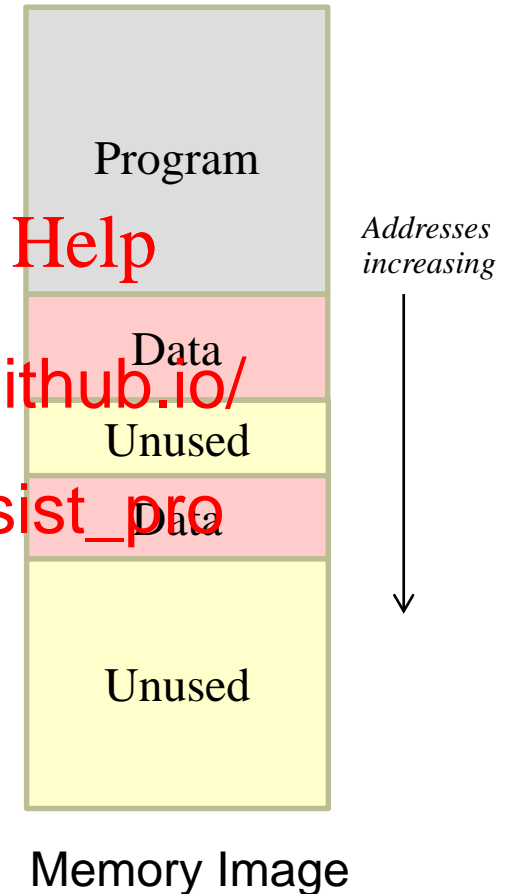
<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Source and Object Programs

- A **source program** is a program written in a HLL or assembly language, input to a compiler or assembler.
- An **object program** is a program translated into machine code by the compiler or assembler
- An assembler or compiler must generate the necessary content for all memory locations required for program or data.

- This is a list of binary **memory image**, which can be stored e.g. on disk and loaded into memory when the object program is executed.
- When instructed the system **loads** the image and sets the PC to the **start address** (this will be recorded as **metadata** along with the image). Execution of the object program then begins.



Assemblers and Compilers

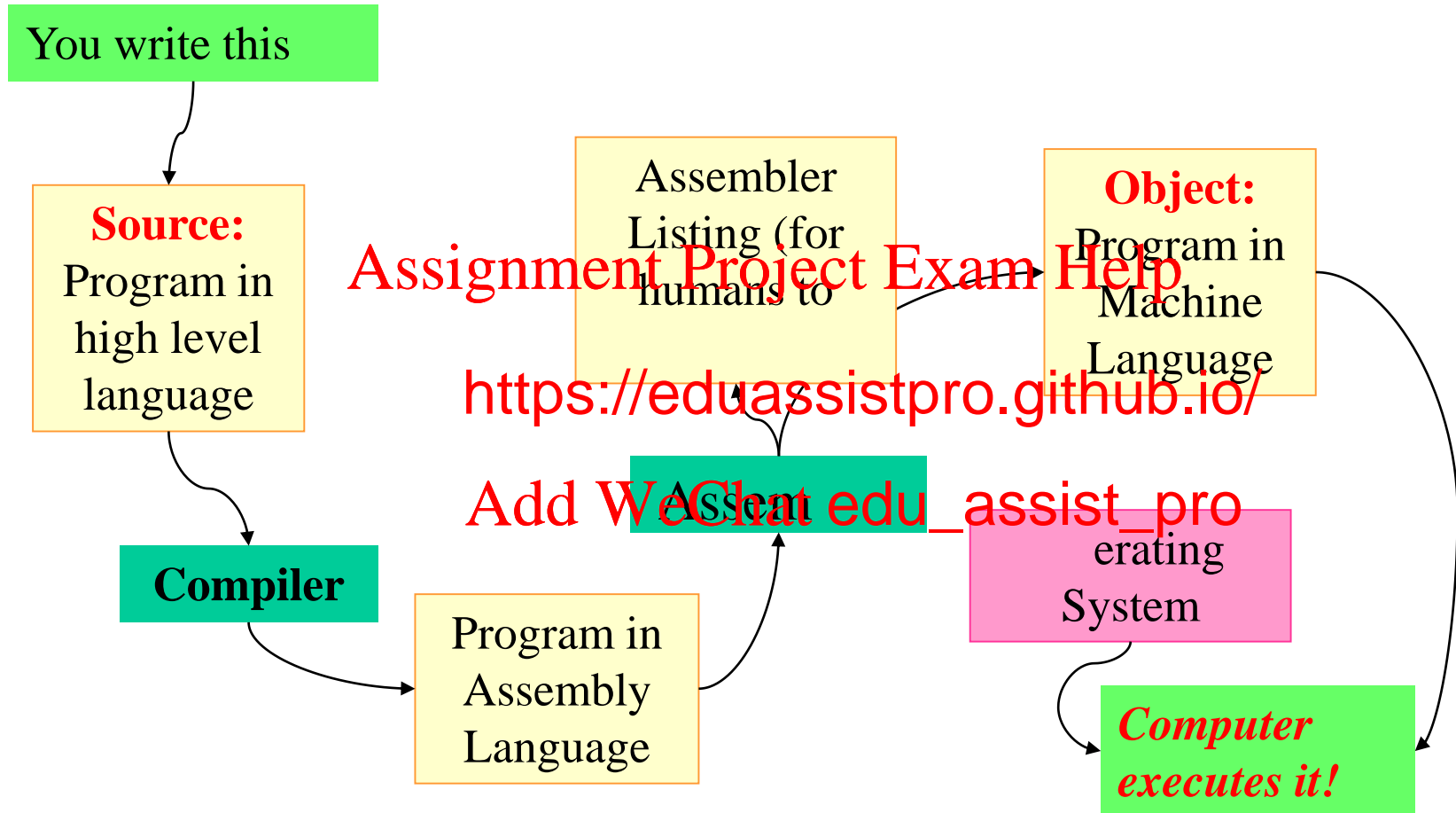
- An assembler generates machine code from the assembly language (*also known as “assembler”*) statements.
- As in a high level language assembly language supports named constants and variables:
 - constants can be incorporated into the machine code itself.
 - variables are memory locations. Assembly language can be labelled (named)
- To generate a loada
 - produce a memory location for instructions in consecutive words
 - keep track of the address of each declared variable
 - generate each instruction's fields as specified by the assembly language statement, inserting the correct op-code in the op-field.
 - assemble the pieces
- A compiler for a HLL targeted at a given CPU architecture will often generate assembly language for that architecture to allow human inspection.
 - The assembler is then run as a second step to generate machine code

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

The Big Picture: HLL programs



How Programs are Executed

- A running computer is always executing instructions. If the CPU stops executing instructions it is said to be in a **halt state** and does nothing.
 - A modern PC is almost never in a halt state unless it is asleep or has crashed.
- To get a CPU out of a halt state it needs to be given an external signal such as a **reset** or **interrupt**. This is done via a hardware signal.
- On a reset for exam d its program counter to some predetermine here it will start executing code call <https://eduassistpro.github.io/>
 - if there is no machine code at the reset ve will crash.
- On most systems the reset handler routin so always present. Usually it begins the loading of the **operating system (OS)**.
- The operating system runs when nothing else is running.
- The simplest operating systems, sometimes called **monitors**, just keep checking some input device (like a keyboard) and wait for a user to enter a text command which is then interpreted.

Execution in Sigma16

- The Sigma16 computer is a simulated or **virtual** computer and it has its own simple operating system as well as a set of user tools.
- These tools allow the user to load an image file into the (simulated) memory and have the (simulated) CPU execute it.
 - On Sigma16 the image is always loaded with the assumption that execution will begin at address
 - There has to be a valid entry point or error will be reported.
 - Even if there is machine code, it must be a valid program or the machine will behave unpredictably and eventually crash.
 - On a real machine, the user would have control over where the execution should begin.
- Execution continues until the CPU encounters a **TRAP R0,R0,R0** instruction which causes it to stop and return control to the Sigma16 OS.

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat: edu_assist_pro

Compiling HLLs

- Every High level language (HLL) statement is converted into assembly language by the compiler.
 - If it can be done by a compiler it can be done by a human.
 - Sometimes a human can do a better job than a compiler but it is a very labour intensive task.
 - It is however a very instructive one.

Assignment Project Exam Help

- Let's look at some examples
- Consider an assignment

<https://eduassistpro.github.io/>

- We start by assuming that variables are stored in memory. They can be set up by DATA statements at the end of the code such as

`a DATA $0000 ;Dec — initialises to 0`

- Since we need to do arithmetic on these variables, we must load them into registers. Any registers will do but let's choose R1, R2 and R3.

```
LOAD     R1,a[R0]     ;R1 = a
LOAD     R2,b[R0]     ;R2 = b
LOAD     R3,c[R0]     ;R3 = c
MUL     R4,R2,R3     ;R4 = b*c
ADD     R4,R1,R4     ;R4 = a+(b*c)
STORE   R4,x[R0]     ;x = a+(b*c)
```

A Complete Program

```

LEA      R1, 3[R0]      ; R1 = 3
LOAD     R2, x[R0]      ; R2 = x
MUL      R3, R1, R2      ; R3 = 3*x
STORE    R3, y[R0]      ; y = 3*x
TRAP     R0, R0, R0      ; terminate
    
```

Comments (;)

Full line comment (;)

```

; Variable declaration:
x      DATA $0021      initial 33
y      DATA $0000      initial 0
    
```

Labels

Assembler directives. Each DATA statement reserves a memory location, gives it a label and initialises it to the value on the right

Assembler Listing

Line	Addr	Code	Source statement
1	0000	f100 0003	LEA R1,3[R0]
2	0002	f201 0008	LOAD R2,x[R0]
3	0004	2312	MUL R3,R1,R2
4	0005	f301 0000	STORE R3,R[R0]
5	0007	d000	TRAP R0,R0,R0
6	0008		n and initialisation
7	0008	0021	
8	0009	0000	y DAT

Addr	Symbol	Def	Usage
0008	x	[6]	[]
0009	y	[7]	[]

Location of word

Object code

Source code

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Statement Translation

- For each type of statement in the HLL, there is a standard implementation technique using machine code/assembler.
- We'll look at 2 methods for translating a full HLL program:

1. *Statement-by-statement style.*

- Every statement in the HLL program is translated in full, all by itself, into a block of Assembly Language instructions.
- Each block of instructions is stored in memory, and it needs from memory, and finishes by storing to memory.
- The HLL statement is used as a full line of code in the block of instructions, and each individual instruction has a comment that it does.
- This is straightforward and clear but can result in some inefficiency.

2. *Register-variable style* is like statement by statement style, except

- Keep commonly used variables in registers
- Make a table showing which register contains which variable & include as a comment
- Omits unnecessary loads & stores, making the program shorter and faster

Example: Statement by Statement Style

x = 50;
y = 2*z;
x = x+1+z;

```
; x = 50;  
LEA      R1,$0032[R0]  ; R1 = 50  
STORE    R1,x[R0]      ; x = 50  
  
; y = 2*z;  
LEA      R1,$0002[R0]  ; R1 = 2  
]        ; R2 = z  
]        ; R3 = 2*z  
]        ; y = 2*z  
  
; x = x+1+z  
LOAD     R1,x[R0]      ; R1 = x  
LEA      R2,$0001[R0]  ; R2 = 1  
LOAD     R3,z[R0]      ; R3 = z  
ADD      R4,R1,R2      ; R4 = x+1  
ADD      R4,R4,R3      ; R4 =x+1+z  
STORE    R4,x[R0]      ; x = x+1+z
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Example: Register Variable Style

```
; Usage of register variables:  
;   R1 = x  
;   R2 = y  
;   R3 = z
```

Assignment Project Exam Help

```
x = 50;  
y = 2*z;  
x = x+1+z;
```

<https://eduassistpro.github.io/>

Add WeChat: edu_assist_pro

```
LEA    R1,$0032[R0] ; x = 50  
        z[R0] ; Get z  
        $0002[R0] ; R4 = 2  
        R4,R3 ; y = 2*z  
        LE    0001[R0] ; R4 = 1  
        AD    ,R4 ; x = x+1  
        ADD    R1,R1,R3 ; x = x+z  
        STORE    R1,x[R0] ; Save x  
        STORE    R2,y[R0] ; Save y
```

An Example Program: Add

```
; Program Add.  y = x+32; initially x = 10
```

```
; The program
```

```
LOAD R1, x; R01, R1 = x
```

```
LEA R2,
```

```
ADD R3,
```

```
STORE R3,
```

```
TRAP R0, R0, R0; Stop
```

```
; The data
```

```
x DATA 10 ; 10
```

```
y DATA 0 ; 00
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Each line of source will produce one or two words of object code. Some words are instructions, others are data.

Use the Sigma 16 environment to generate an assembler listing for this program

Practice exercise

- Translate this code fragment to assembly language:

x = 13;

y = x + 2;

z = x - (y

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Assembler Listing

When you run the assembler, and give it program Add as input, it will print a listing like this...

Line	Address	Machine code	Program
0			; Program Add
1			; y = x+32; initially x = 10
2			
3	0000	F101 0	; R1 = x
4	0002	F200 0	; R2 = 32
5	0004	0312	ADD ; R3 = x+32
6	0005	F302 0009	STORE ; y = x+32
7	0007	d000	TRAP R0,R0,R0 ; Stop
8	0008	000a	x DATA \$000a ; 10
9	0009	00ab	y DATA \$00ab ; 171

Location of word

Object code

Source code