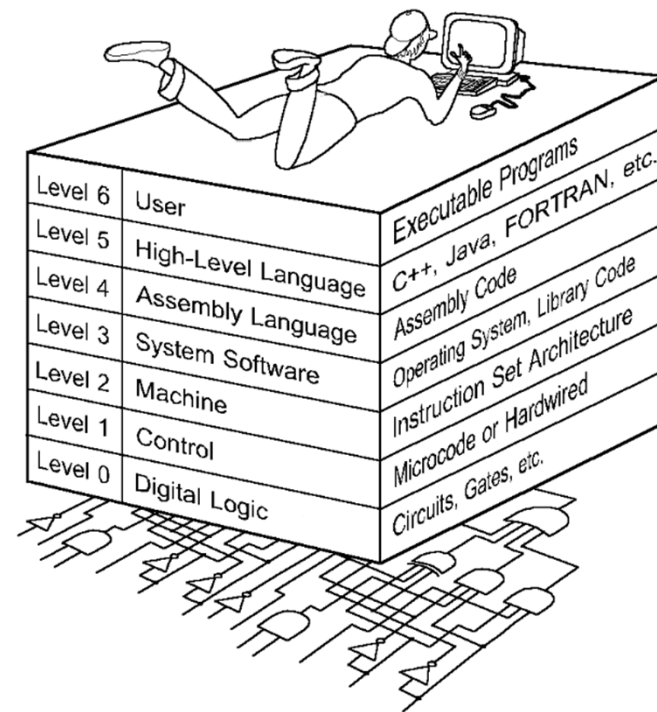


Simplified IAS language



The Computer Level Hierarchy

- Each virtual machine layer is an abstraction of the level below it.
- The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.
- Computer circuits ultimately carry out the work.



The Computer Level Hierarchy

- Level 6: The User Level
 - Program execution and user interface level.
 - The level with which we are most familiar.
- Level 5: High-Level Language Level
 - The level with which we interact when we write programs in languages such as C, Pascal, Lisp, and Java.

The Computer Level Hierarchy

- Level 4: Assembly Language Level
 - Acts upon assembly language produced from Level 5, as well as instructions programmed directly at this level.
- Level 3: System Software Level
 - Controls executing processes on the system.
 - Protects system resources.
 - Assembly language instructions often pass through Level 3 without modification.

The Computer Level Hierarchy

- Level 2: Machine Level
 - Also known as the Instruction Set Architecture (ISA) Level.
 - Consists of instructions that are particular to the architecture of the machine.
 - Programs written in machine language need no compilers, interpreters, or assemblers.

The Computer Level Hierarchy

- Level 1: Control Level
 - A *control unit* decodes and executes instructions and moves data through the system.
 - Control units can be *microprogrammed* or *hardwired*.
 - A microprogram is a program written in a low-level language that is implemented by the hardware.
 - Hardwired control units consist of hardware that directly executes machine instructions.

The Computer Level Hierarchy

- Level 0: Digital Logic Level
 - This level is where we find digital circuits (the chips).
 - Digital circuits consist of gates and wires.
 - These components implement the mathematical logic of all other levels.

IAS Hardware

- There are 4 registers
 - PC = Program Counter
 - IR = Instruction Register
 - AC = Accumulator Register
 - MQ = Multiplier Quotient Register
 - Used in Multiplication and division.

Uses the Fetch-Execute Cycle

1. Read in an instruction from Memory listed by the PC
2. Increment the PC by 1
3. Decode the instruction
4. Execute the instruction
5. Goto step 1

Example 1

main () {	0 15 a
int a=15;	1 5 b
int b=5;	2 0 c
int c;	3 begin
c = a + b;	4 .c = a +b
}	5 load M(0)
	6 add M(1)
	7 stor M(2)
	8 halt

Example 2

```
main () {  
    int a,b,c;  
    a = 15;  
    b = 5;  
    c = a + b;  
}
```

Note:

- 15 and 5 are constants needed in the program.
- Variable a, b, and c must be initialized in the program.

```
0 15  
1 5  
2 a  
3 b  
4 c  
5 begin  
6 . a = 15  
7 load M(0)  
8 stor M(2)  
9 . b = 5  
10 load M(1)  
11 stor M(3)  
12 . c = a + b  
13 load M(2)  
14 add M(3)  
15 stor M(4)  
16 halt
```

Example 2 (continued)

Or you could optimize
the code a little.

NOTE FOR
HOMEWORK.

You do not need to
optimize, but it
must be at least as
efficient as the
“original” code.

```
0 15
1 5
2 a
3 b
4 c
5 begin
6 . a = 15
7 load M(0)
8 stor M(2)
9 . b = 5
10 load M(1)
11 stor M(3)
12 . c = a + b
13 add M(2)
14 stor M(4)
15 halt
```

Example 3

```
main () {  
  int a=15, b=5, c;  
  if (a >= b)  
    c = a - b;  
  else  
    c = a + b;  
}
```

```
0 15  a  
1 5   b  
2 c  
3 begin  
4 . If (a >=b)  
4 load M(0)  
5 sub M(1)  
6 jump+ M(8)  
7 jump M(12)  
8 .true, c=a-b  
8 load M(0)  
9 sub M(1)  
10 stor M(2)  
11 jump M(15)  
12 .false c = a+b  
12 load M(0)  
13 add M(1)  
14 stor M(2)  
15 halt
```

Example 3 (continued)

- Optimized

```
0 15    a
1 5      b
2 c
3 begin
4 load M(0)
5 sub M(1)
6 jump+ M(9)
7 load M(0)
8 add M(1)
9 stor M(2)
10 halt
```

Example3 (with a > b)

main () {	0 15 a
int a=15, b=5, c;	1 5 b
if (a > b)	2 c
c = a – b;	3 1
else	4 begin
c = a + b;	5 . a > b
}	5 load M(0)
	6 sub M(1)
	7 sub M(3)
	8 jump+ M(10)
	9 jump M(14)
	10 . True, c = a- b
	10 load M(0)
	11 sub M(1)
	12 stor M(2)
	13 jump M(17)
	14 . False, c = a + b
	14 load M(0)
	15 add M(1)
	16 stor M(2)
	17 halt

Example 4

```
main () {  
    int x=0;  
    while (x < 5) {  
        x=x+1;//or ++x;  
    }  
}
```

A NOTE FOR HOMEWORK

You **CAN NOT** change a
top testing loop to a
bottom testing loop!

```
0 5  
1 1  
2 0      x  
3 begin  
4 . While (x<5)  
4 load M(2)  
5 sub M(0)  
6 jump+ M(11)  
7 . X=x+1  
7 load M(2)  
8 add M(1)  
9 stor M(2)  
10 jump M(4)  
11 halt
```

Example 4 (continued)

What if

while ($x \leq 5$)

Could change the 5 to a x
< 6

Or

```
0 5
1 1
2 0          x
3 begin
4 . While ( $x \leq 5$ )
4 load M(2)
5 sub M(0)
6 sub M(1)
7 jump+ M(12)
8 .  $x = x + 1$ 
8 load M(2)
9 add M(1)
10 stor M(2)
11 jump M(4)
12 halt
```

Example 5

```
main () {  
    int l , x = 0;  
    for (l =1;l<=6; ++l) {  
        if (l % 2==0) {  
            // % is mod, so l mod 2  
            x = x +l;  
        }  
    }  
    x = x * 2;  
}
```

Code on next side.

Example 5 (continued)

0 7	15 div M(2)
1 1	16 sub M(1)
2 2	17 jump+ M(23)
3 i	18 . x +=l;
4 0 x	19 load M(4)
5 begin	20 add M(3)
6 . i =1	21 stor M(4)
7 load M(1)	22 . ++l
8 stor M(3)	23 load M(3)
9 . l<=6	24 add M(1)
10 load M(3)	25 stor M(3)
11 sub M(0)	26 jump M(10)
12 jump+ M(27)	27 . x = x *2
13 . If (l%2)	27 load MQ,M(4)
14 load M(3)	28 mul M(2)
	29 stor M(4)
	30 halt

Example 6

```
main () {  
  int a=2, b=2, l;  
  l = 1;  
  while (l < 10) {  
    a = a +b;  
    l = l +1;  
  }  
}
```

Give it a try.

Example 6 (continued)

main () {	0 1		
int a=2, b=2, I;	1 10		
I = 1;	2 2	a	
while (I < 10) {	3 2	b	15 add M(3)
a = a +b;	4 i		16 stor M(2)
I = I +1;	5 begin		17 . I=I+1
}	6 . I =1		18 load M(4)
}	7 load M(0)		19 add M(0)
	8 stor M(4)		20 stor M(4)
	9 . while (I < 10)		21 jump M(10)
	10 load M(4)		22 halt
	11 sub M(1)		
	12 jump+ M(22)		
	13 . a = a +b		
	14 load M(2)		

Simulator notes.

- Variables before the begin can have comments after the value
 - After the begin, comments after a statement will cause a compile error.
- You can use the same line again
 - The simulator will overwrite the memory space.
 - 27 .x =x *2
 - 27 load M(1)
 - In memory at 27 will be load M(1)
- The simulator is not case sensitive.
 - load m(1) same as Load M(1)
- YES you have to use line numbers.
 - But you can leave out a line number
 - If that line number is executed, then the nop assembly command will be executed.

Helpful things to know

- What is the logic to figure out the following
- If $(a \geq b)$ or $(a \geq 1)$ or any constant
- If $(a > b)$ or $(a > 1)$
- If $(a < b)$ or $(1 < b)$
- If $(a \leq b)$ or $(1 \leq b)$
- If $(a == 1)$ or $(a == b)$
- Complex if statements
 - If $((a > 10) \ \&\& \ (b < 10))$

Helpful things to know (2)

- What would the general structure of the code before for this if statement

```
If (a > 4) {  
    ...  
} else if ( a> 0) {  
    ...  
} else {  
    ...  
}
```

Helpful things to know (3)

- What would the general structure of the code before for this code

```
for (i=1; i<10; ++i) {  
    if (i>5) {  
        ...  
    } else {  
        ...  
    }  
}
```

- Does the assembly code look any different for a while loop, instead of for loop?