Lecture - 15 (Continue) | <8/14/2019>

- Stack operation
- Memory Segmentation

Example - High level Language execution :-

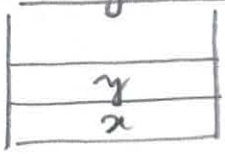
C - Like program :-

```
int mul (int x, int y) {
    int prod = 0;
    for (int i = y; i != 0; i++)
        prod = prod + x;
    return prod;
}
```

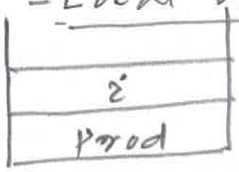
variables :- parameters :- x, y.
local variables :- prod, i
Constant :- 0, 1

Memory segment :-

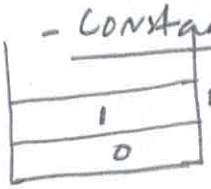
- Arguments -



- Local variables -



- Constant -

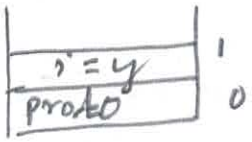


(2)

Corresponding VM-code :-

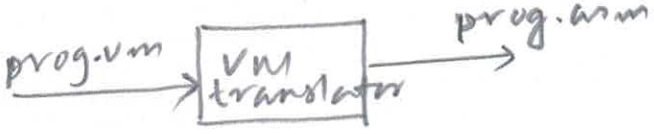
```
prod=0   push constant 0
         pop local 0
i=y      push argument 1
         pop local 1
```

stack




```
Label loop
        push constant 0
        push local 1
        eq
        if-goto end          // i!=0;
        push local 0
        push argument 0
        add
        pop local 0          // prod = prod+x;
        push local 1
        push constant 1
        sub
        pop local 1          // i=i-1;
        goto loop
        ~~Label end~~
Label end
        push local 0
        return               // return prod;
```

Next task :- translate this VM code to assembly code



- Mapping (stack & memory segment) ✓
- command to instruction translation ✓

Memory segment :-

s1 - argument

s2 - Local

s3 - static

s4 - constant

s5 - this, that (Heap)

s6 - pointer to (this / that)

s7 - temp

to be mapped to the physical memory RAM (Data memory)

(in this case ROM is not needed (the instruction memory))

Review of the RAM allocation / space :-

RAM address

0	—	SP (R0) → alternative name.
1	—	LCL (R1)
2	—	ARG1 (R2)
3	—	THIS (R3)
4	—	THAT (R4)
0-15	—	R0 - R15

16384	—	SCREEN
24576	—	KBD

(4)

- The RAM space :-

0 - 15 —— [0 - 4] · RAM[0] → SP
RAM[1] → LCL
RAM[2] - A
RAM[3] - TH
RAM[4] - TH

16 - 255 —— Static variable (segment)

256 - 2047 —— Stack manipulation

2048 - 16383 —— Heap (used to store array, struct obj)

16384 - 24575 — Memory mapped I/O

24577 - 32767 — unused memory space

—

Segment mapping

- Local segment —— base address → LCL
- argument segment — base addr → ARG
- this segment — base addr → THIS
- that segment — base addr → THAT

↓

The memory space can be allocated from either stack, Heap, or unused free space

General practice → Local & argument are allocated in a stack region

this & that are allocated in Heap region

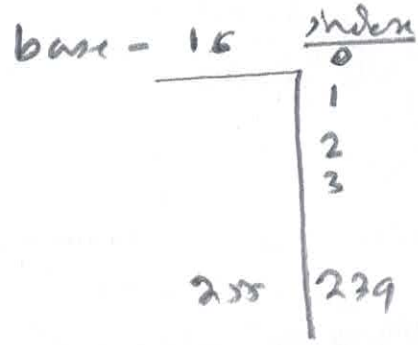
pointer and temp :- pointer gets dedicated RAM[3] & RAM[4] locations

& temporary gets dedicated RAM[5 —— 12] locations

Constant :- In our hv assembly language constant is provided as immediate data. Therefore constant does not require any special physical location.

Static - A global segment for a given program. prog.vm

Dedicated location - 16 - 255



Example

prog.vm :-

variable - v_1, v_2, v_3, v_4

0 1 2 3

prog.0 → v_1 — 16

prog.1 → v_2 — 17

prog.2 → v_3 — 18

Prog.3 → v_4 — 19

For different program **my.vm** :—

my.0

my.1

my.2

my.3

⑥ **Command to instruction translation :-**

Example

pop segment index → +ve integer in range $(0 \text{ to } 2^{15}-1)$

push segment index

- argument
- local
- static
- constant
- this
- that
- pointer
- temp

Push constant 6

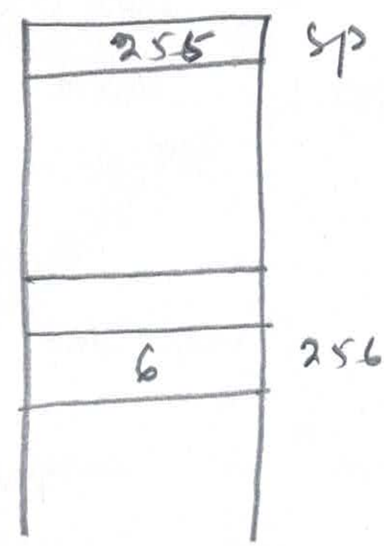
*sp ← 6 ✓

sp ← sp+1

H-asm equivalent

*sp←6:

```
@6      // A←6
D=A     // D←6
@sp
A=M
M=D
@sp
M=M-1   sp←sp+1
```



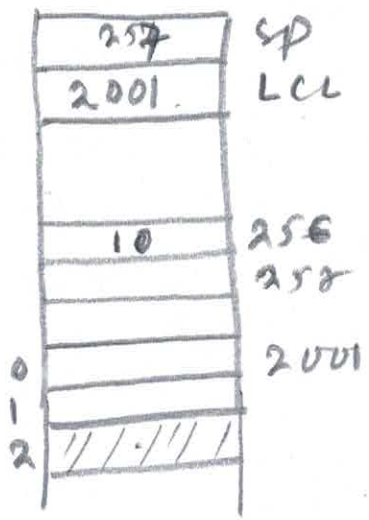
(Design - Stack address stored

In general

push constant i

*sp ← i, sp ← sp-1

pop local 2



addr ← LCL + 2
sp ← sp - 1
*addr ← *sp

H-asm equivalent :-

@LCL
D = M
@2
D = D + A
@sp @2000, M = D
A = M
A = A - 1
D = M
@2000
A = M
M = D

Similarly, all the push and pop with different segment can be translated to asm equivalent.

— Translating arithmetic & logic command :-

add → D+A or D+M as per the location of operand

(8) Implementation.

get operand 1

get operand 2

add 1 and 2

put the result on top of stack