ASSEMBLY LANGUAGE

13B - Wednesday 25th January 2023

1

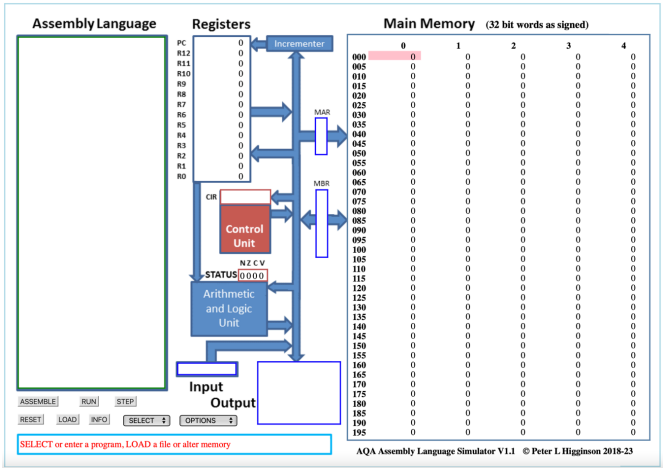
WHY ARE WE EVEN LEARNING THIS?
WHY CAN'T WE JUST USE PYTHON?
(OR WHATEVER YOUR FAVORITE LANGUAGE IS)

- In reality we don't write programs in it very often*
- But we need to know it:
  - Finding Bugs
  - Incredible Performance
  - System Software
  - Malware

*IT'S IN YOUR EXAM

AND IT WILL COME UP

2



3

RECAP OF INSTRUCTIONS

LDR Rd, <memory ref>	Load the value stored in the memory location specified by <memory ref> into register d.
STR Rd, <memory ref>	Store the value that is in register d into the memory location specified by <memory ref>.
ADD Rd, Rn, <operand2>	Add the value specified in <operand2> to the value in register n and store the result in register d.
SUB Rd, Rn, <operand2>	Subtract the value specified by <operand2> from the value in register n and store the result in register d.
MOV Rd, <operand2>	Copy the value specified by <operand2> into register d.
CMP Rn, <operand2>	Compare the value stored in register n with the value specified by <operand2>.
B <label>	Always branch to the instruction at position <label> in the program.
B<condition> <label>	Branch to the instruction at position <label> if the last comparison met the criterion specified by <condition>. Possible values for <condition> and their meanings are: EQ: equal to NE: not equal to GT: greater than LT: less than
AND Rd, Rn, <operand2>	Perform a bitwise logical AND operation between the value in register n and the value specified by <operand2> and store the result in register d.
ORR Rd, Rn, <operand2>	Perform a bitwise logical OR operation between the value in register n and the value specified by <operand2> and store the result in register d.

EOR Rd, Rn, <operand2>	Perform a bitwise logical XOR (exclusive or) operation between the value in register n and the value specified by <operand2> and store the result in register d.
MVN Rd, <operand2>	Perform a bitwise logical NOT operation on the value specified by <operand2> and store the result in register d.
LSL Rd, Rn, <operand2>	Logically shift left the value stored in register n by the number of bits specified by <operand2> and store the result in register d.
LSR Rd, Rn, <operand2>	Logically shift right the value stored in register n by the number of bits specified by <operand2> and store the result in register d.
HALT	Stops the execution of the program.

4

IF STATEMENTS

```
if ...:
  #INNER CODE
 # After Code

CMP ...
B<NOT Condition> label
 # Inner Code
label:
 # After
```

IF ELSE

```
if <Condition>:
  # TRUE Code
else:
  # FALSE Code
# After Code
```

5

YOUR TURN

```
x = 0
if y == 7:
  x = 100
else:
  x = 10

Use R0 for x
Use R1 for y
```

6

WHILE LOOPS

```
while ...:
  # INNER CODE

loopStart:
 CMP ...
 B<NOT Condition> loopDone
 # Inner Code
 B loopStart
loopDone:
 # After
```

```
while ...:
  # INNER CODE

 B test
loopTop:
 # Inner Code
test:
 CMP …
 B<Condition> loopTop
 # After Code
```

7

YOUR TURN

```
sum = 0
while x > 1:
  sum = sum + x
  x = x - 1
```

Store the result of sum in location 42.
Assume R0 contains the value x
You may use R1 for other steps

8

YOUR EXAM

Machine-code/assembly language operations

Content

Understand and apply the basic machine-code operations of:

- load
- add
- subtract
- store
- branching (conditional and unconditional)
- compare
- logical bitwise operators (AND, OR, NOT, XOR)
- logical
  - shift right
  - shift left
- halt.

Use the basic machine-code operations above when machine-code instructions are expressed in mnemonic form-assembly language, using immediate and direct addressing.

Types of Question:

- Values in Registers + Tracing.

- Write small assembly programs.

- Operand vs Opcode.

9

TIPS FOR SOLVING QUESTIONS

- Treat each variable as a register.
- Treat each register as a variable.
- Draw arrows on Assembly Code.
- Memorise the common If and While structure.
- Think about your code in a language you're used to.
- Practice, Practice, Practice.

BUBBLE SORT

```
data = [2, 47, 4, 23, 9, 11]
n = 6
while True:
    current_index = 0
    num_swaps = 0
    while n >= (current_index + 2):
        next_index = current_index + 1
        current = data[current_index]
        next = data[next_index]
        if current > next: # So we should swap
            data[current_index] = next
            data[next_index] = current
            num_swaps = num_swaps + 1
        current_index = next_index
    if num_swaps == 0:
        break # Otherwise do the loop again
```

10