4.7.3.5 Assembly Language

AQA Assembly Instruction Set

This is the AQA instruction set that will be given to you in the paper 2 exam.

The instructions may use immediate or direct addressing.

Add Rd, Rn, #25

- This instruction adds the value in the register Rn to the decimal value 25 and stores the result in register Rd. As the operand 25 is the data, the addressing mode being used is immediate addressing.

Sub Rd, Rn, R1

- This instruction subtracts the value in the register R1 from the value in the register Rn and stores it in the register Rd. As the operand R1 is an address, the addressing mode being used is direct addressing.

`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.

Assembly Tips: Loops

When looping in assembly language create labels at the start and end of the loop section. Check a condition that would end the loop and if it should then branch the end label. At the end of the loop section, branch to the start

```
int x = 0
while (x < 5) {
  x = x + 1
}
```

```
  MOV R1 #0
loopstart:
  CMP R1, #4
  BGT loopend
  ADD R1, R1, #1
  B loopstart
loopend:
  HALT
```

The loop exits when R1 becomes 5, so the comparison checks whether R1 is greater than 4.

Assembly Tips: Ifs

When implementing an IF/ELSE statement in assembly, it is often useful to branch to one section of code when the condition is true and allow the other section to execute by default.

```
int x = 0
if (x > 5) {
  x = x - 1
}
else {
  x = x + 1
}
```

```
  MOV R1 #0
  CMP R1, #5
  BGT lessthan5
  ADD R1, R1, #1
  B end
lessthan5:
  SUB R1, R1, #1
end:
  HALT
```