Comp Science Assembly 2208 - Midterm #2 Review

(Chapter 3)

- CPU = reads instructions from memory and executes them.

- Registers = store temporary data for intermediate calculation results.

- Program Counter (PC) = is the register that contains the address of the next instruction to be executed.

- Condition Code Register (CCR) = is a collection of flag bits for a processor.

- Instruction Format = *operation* register_destination, register_source1, register_source2.

- Stored Program Machine = a computer that has a program in digital form in its main memory.
    - The program and data are stored in the same memory.
    - Operates in a fetch/execute mode.
    - Modern computers are pipelined = fetch and execution operations overlap.

- Instruction register (IR) = stores the instruction most recently read from main memory, and makes it the instruction currently being executed.

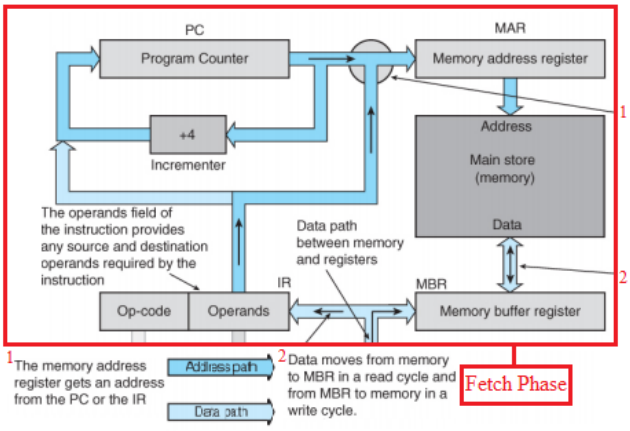
- Memory Address Register (MAR) = stores the address of the location in main memory that is currently being accessed with read or write.

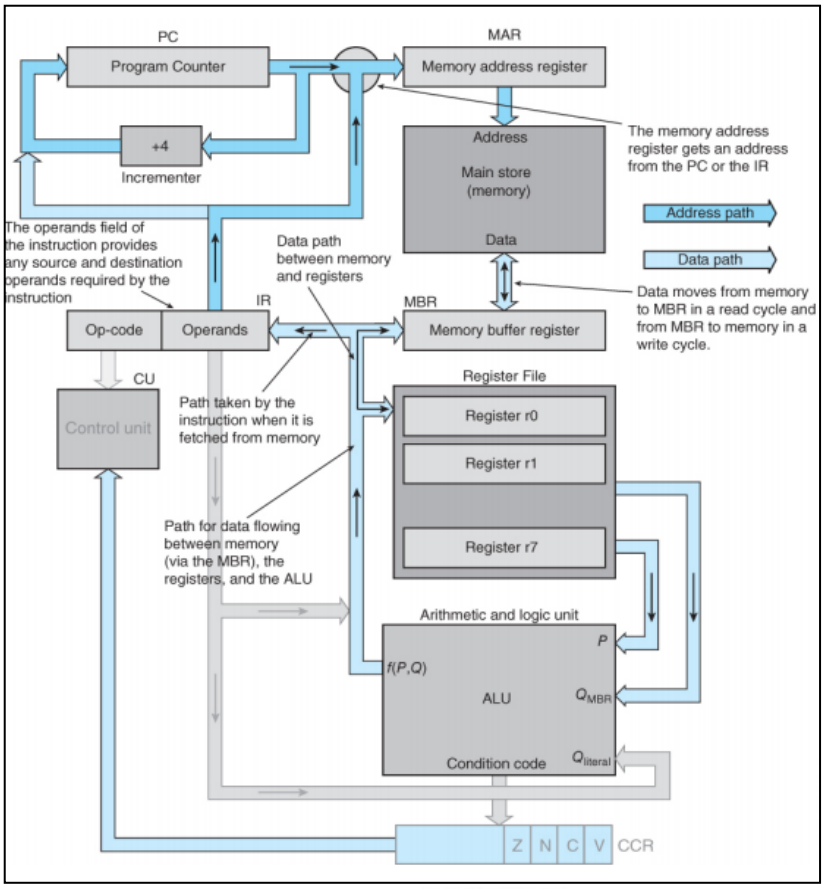
- Memory Buffer Register (MBR) = stores the data that has just been read from main memory, or data to be immediately written to main memory.

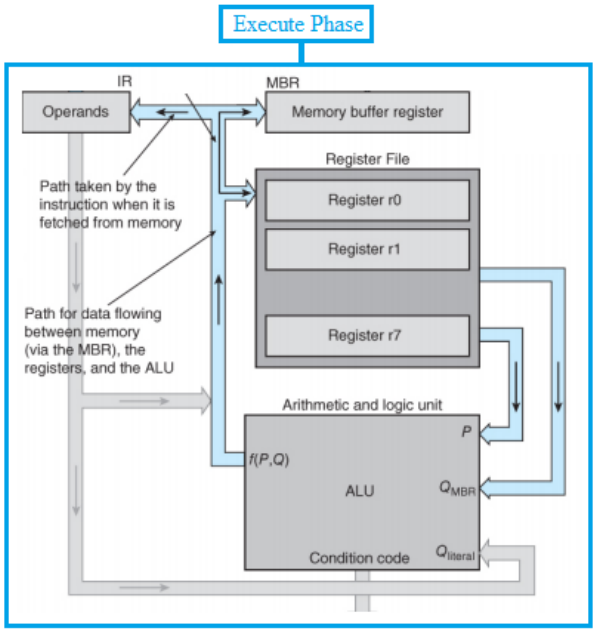
- *Fetch Phase:*
    - The PC supplies the next instruction to be executed to the MAR to read the instruction.
    - Then the PC is incremented by the size of an instruction.
    - The instruction is read and loaded into the MBR and copied to the IR where the op-code is decoded.

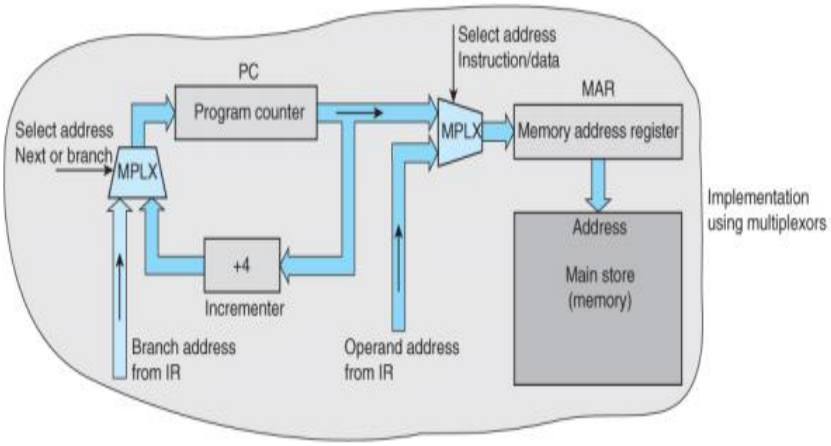
- *Execution Phase:*
    - Operands are read from the register file and transferred to the ALU.

- The ALU operates on them and passes the result to the destination register.










```
FETCH   [MAR] ← [PC]          ;copy PC to MAR
        [PC] ← [PC] + 4       ;increment PC
        [MBR] ← [[MAR]]       ;read instruction pointed at by MAR
        [IR] ← [MBR]          ;copy instruction in MBR to IR

LDR     [MAR] ← [IR(address)] ;copy operand address from IR to MAR
        [MBR] ← [[MAR]]       ;read operand value from memory
        [r1] ← [MBR]          ;add the operand to register r1
```

- *Dealing with Constants* – a constant when added for example, is routed from the operand field of the IR rather than the MBR.

Common Instructions	
LDR **r0,**address	***Load*** the contents of the memory location at address into register **r0**.
STR r0,**address**	***Store*** the contents of register r0 at the specified **address** in memory.
ADD **r0**,r1,r2	***Add*** the contents of register r1 to the contents of register r2 and store the result in register **r0**.
SUB **r0**,r1,r2	***Subtract*** the contents of register r2 from the contents of register r1 and store the result in register **r0**.
BPL target	***If*** the result of the previous operation was ***plus*** **(*+ve*** or ***zero*)** ***then*** branch to the instruction at address target.
BEQ target	***If*** the result of the previous operation was ***zero***, ***then*** branch to the instruction at address target.
B target	***Branch unconditionally*** to the instruction stored at the memory address target.
Note the number of operands in each instruction.	

- Flow Control = any action that modifies the instruction-by-instruction sequence of a program.

- Conditional Behavior = allows a processor to select one of two possible options.
    - Example – BEQ is a conditional instruction. Either the program continues normally or branches.

- Status Bits (Flags) = condition or status information.

- Condition Code Register (CCR) = when a computer performs an operation, it stores it the status bits here.
    - It records Zero (Z), Negative in 2's complement (N), Carry (C), oVerflow (V).

- Complex Instruction Set Computer (CISC) = update status flags after each operation.
    - Example – Intel IA32.
    - Allow memory-to-register and register-to-memory processing.\

- Typically use two-address instructions, where one is memory, and one is register.

- Reduced Instruction Set Computer (RISC) = requires the programmer to request updating the flags.
  - Example – ARM.
    - ARM does this by appending an S to the instruction like SUBS or ADDS.
  - Only allows for register-to-register processing, and use a special LOAD and STORE for memory-to-register and vice versa transfers.
  - Typically use three-address processing where all three are registers.

Example of a Conditional Program

```
        MOV  r0,#1      ;Put 1 in register r0 (the counter)
        MOV  r1,#0      ;Put 0 in register r1 (the sum)
Next    ADD  r1,r1,r0   ;REPEAT: Add current counter to sum
        ADD  r0,r0,#1   ;    Add 1 to the counter
        CMP  r0,#21     ;    Have we added all 20 numbers?
        BNE  Next       ;UNTIL we have made 20 iterations
        STOP            ;If we have then stop
```

- *ARM processors* have general-purpose registers and 2 special-purpose registers (can't be used for general data processing).

- Literal/ Immediate = the actual value is part of the instruction.

- Direct/Absolute = the instruction proves the memory address of the operand (ARM doesn't support this).

- Register Indirect/Pointer Based/Indexed = a register contains the address of the operand.

- *Two-addressing* results in overwriting of one of the source operands.

- *One-address* processing requires using a fixed register, called an accumulator.
  - Example – ADD #1, adds 1 to the accumulator.

- *Zero-addressing* is when the machine operates on data at the top of a stack, and a pure zero-addressing machine is impractical.
  - Although they handle Boolean logic easily.

- *ARM:*
  - Stands for Advanced RISC Machines.
  - Typically used in mobile devices.
  - Started off with 8-bit microprocessors and then moved to 32-bit.
  - In ARM terms, a half-word = 16 bits, a word = 32 bits.
  - Operand values are 32 bits wide, but some instructions generate 64-bit products stored in 2 separate 32-bit registers.
  - It's processor has 16 registers:
    - r0 - r12 are general purpose.
    - r13 is used as a stack pointer (by programmer).
    - r14 is the link register (hardware enforced).
    - r15 is the program counter (hardware enforced).

- Assembler Directive = AREA name, CODE, READONLY and ENTRY and END.

- DCD constant {, constant} = sets up a 32-bit constant in memory.
  - An'&' is also synonymous for DCD.

- DCW constant {, constant} = sets up a 16-bit constant in memory.

- DCB constant {, constant} = sets up an 8-bit constant in memory.
  - An'=' is also synonymous for DCB.

- Pseudo Instructions = is an operation the programmer can use when writing code, where the assembler generates suitable code to carry out the same action.
  - Example – LDR r0 = 0x12345678
  - Example – ADR r1, label ; register 1 points to label.

- Program Counter Relative Addressing:
  - LDR r0, [r1] ; specifies that the operand address is in r1.
  - LDR r0, [r1, #3] ; specifies the operand address is 3 bytes into r1.
  - ARM's PC is typically 8 bytes from the current instruction to be executed.

ARM's Data-Processing Instructions (Arithmetic Instructions)

Addition	ADD
Subtraction	SUB
Negation	NEG
Comparison	CMP
Multiplication	MUL
Bitwise logic operations	AND, OR, EOR
Shift operations	LSL, LSR, ASR, ROR, RRX

- ADC = ADD with carrying.

- RSB = reverse subtraction, so RSB r1, #5, #3 is 5-3.
    - This is useful as with SUB, you can't start with a constant.

- NEG r1, r1 = essentially multiplies the value in r1 by -1.

- MOV = copies the value of an operand into the other operand, it doesn't move it.

- MVN = same as MOV but it also flips the 0's to 1's and the 1's to 0's.

- Implicit Instructions = evaluate an expression and store the result.
    - Example – SUBS r1, r1, #1.

- Explicit instructions = evaluate an expression without storing the result.
    - Example - CMP, r1, r2.

- MUL = multiplies 2 operands together.
    - However it cannot use the same register for a destination and an operand.
    - Additionally, multiplication by a constant is not allowed.
    - All 32 by 32 bit multiplication is truncated to lower-order 32 bits.

- MLA = multiply and accumulate, which performs a multiplication and adds the product to a running total.
    - In the format of MLA Rd, Rm, Rs, Rn ; [Rd] = [Rm] x [Rs] + [Rn].
    - Same rules as MUL.
    - Used for dot products.

- UMLL = Unsigned long multiply.

- UMLAL = Unsigned long multiply-accumulate.

- SMULL = Signed long multiply.

- SMLAL = Signed long multiply-accumulate.

- Logical operations / Bitwise Operations = operations applied to individual bits of a register.

❑ Example: suppose that
- register r0 contains the 8 bits bbbbbbxx,
- register r1 contains the 8 bits bbbyyybb and
- register r2 contains the 8 bits zzzbbbbb,

where
- x, y, and z represent the bits of desired fields and
- the b's are unwanted bits.

```
AND  r0,r0,#2_11          ;Mask r0 to two bits xx
AND  r1,r1,#2_11100       ;Mask r1 to three bits yyy
AND  r2,r2,#2_11100000    ;Mask r2 to three bits zzz
ORR  r0,r0,r1             ;Merge r1 and r0 to get 000yyyxx
ORR  r0,r0,r2             ;Merge r2 and r0 to get zzzyyyxx
```

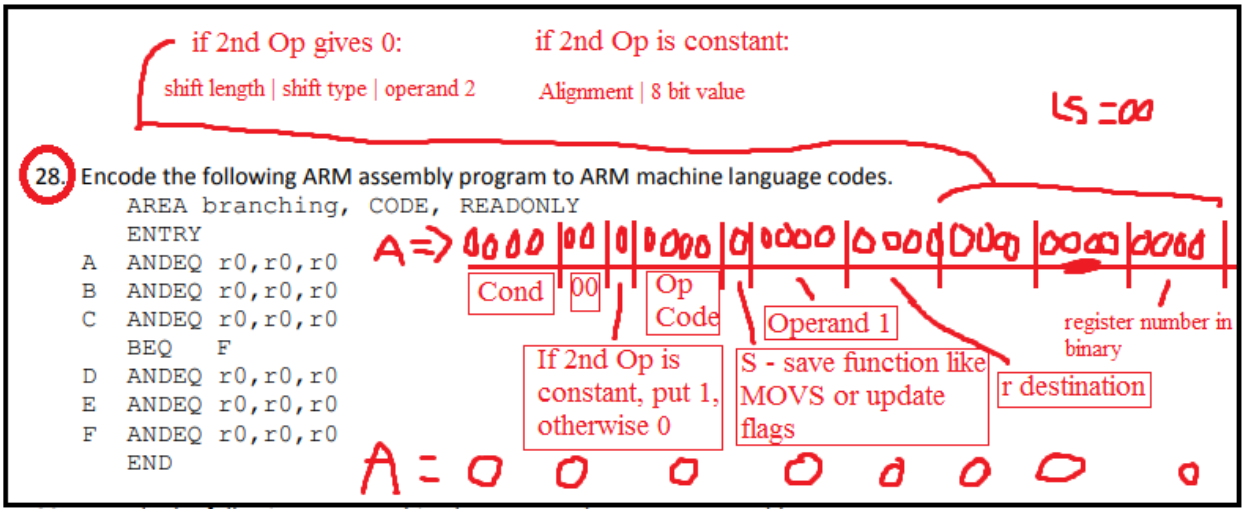
- AND = essentially removes the bits specified as 0 or not specified at all.
- ORR = set's/merges specified bits.
- EOR = toggles the bits specified.
- BIC = bit clear instruction, which AND's its first operand with the complement of its second operand.
  - Example – BIC r0, r1, r2 ; takes op1 first half and replaces rest with 0.
- LSL = logical shift left, add 0's to the back for the amount of shift chosen, then remove the pushed bits in the front. Shift into new 4's after.
- LSR = logical shift right, add 0's to the front for the amount of shift chosen, then remove the pushed bits in the back. Shift into new 4's after.
- ASL = arithmetic shift left, shifts everything to the left and then replaces with 0's.
- ASR = arithmetic shift right, shifts everything to the right and then replaces with 0's.
- ROR = rotate right, takes the front n bits and moves them to the back, no 0 replacement.
- RRX = rotate right through carry.

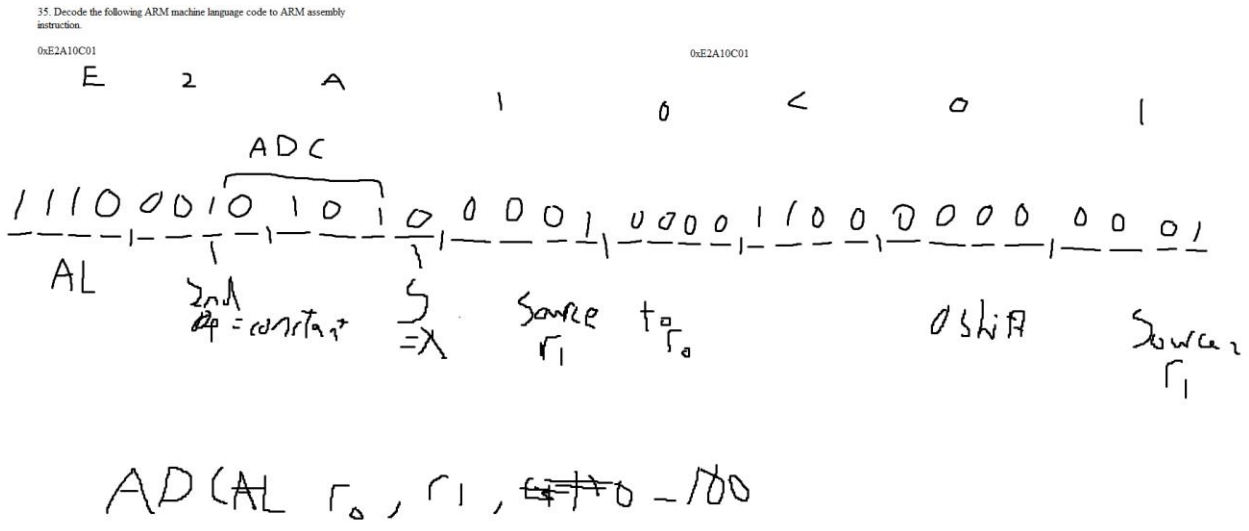
❑ Consider the following if statement,

IF (X == Y)
THEN Y = Y + 1
ELSE Y = Y + 2

```
      CMP r1,r2     ;Compare r1 and r2,
                    ;where r1 contains y and r2 contains x
      BNE Plus2     ;if not equal then branch to the else part
      ADD r1,r1,#1  ;if equal fall through to here
                    ;and add one to y
      B   leave     ;now skip past the else part
Plus2 ADD r1,r1,#2  ;ELSE part add 2 to y
leave  …            ;continue from here
```

- Encoding:
  - Condition, 3 0's, Op-Code, register source (first operand),





ADCAL r0, r1, #0x300