28.4 Instruction Execution Algorithm (Fetch/Execute Cycle)

The functional specification of any CPU is given by how the instruction execution algorithm is to be implemented. This requires the information provided by the instruction set architecture of the instruction set for which the CPU is being designed.

The following algorithm is obtained using the physical instruction formats and semantics of the example instruction set architecture determined above:

1. **INITIALIZE**: This step specifies the address of the first byte of the first instruction of the machine language program.
   PC ← <start address>

2. **FETCH INSTRUCTION**: This step fetch 1, 2, 3, or 4 bytes of memory depending on the size of the instruction as determined from the 1st digit of the opcode.
   IC:IF ← M[PC]
   PC ← PC + 1;
   Case IC is

   When 4 ⇒

   (a) rA:rB ← M[PC];

   (b) PC ← PC + 1;

   (c) valC(7:0) ← M[PC]

   (d) PC ← PC + 1;

   (e) valC(15:8) ← M[PC];

   (f) PC ← PC + 1;

   When 6 or 8 ⇒

   (a) rA:rB ← M[PC];

   (b) PC ← PC + 1;

   When 7 ⇒

   (a) valC(7:0) ← M[PC];

   (b) PC ← PC + 1;

   (c) valC(15:8) ← M[PC];

   (d) PC ← PC + 1;

   When 9 ⇒ PC ← R[rsp]

   When others ⇒ STOP: invalid opcode

3. **DECODE INSTRUCTION**: The step retrieves two values from internal memory as potential operands. Note that either of both of these may in fact not be required to execute a specific instruction.

   valA ← R[rA];

   valB ← R[rB];

4. **EXECUTE INSTRUCTION**: The step performs any computations specified by the instruction.

   Case IC is

   When 4 ⇒
   Case IF is

   When 0 ⇒ valE ← 0 + valC;

   When 1 ⇒ valE ← valB + valC;

   When 2 ⇒ valE ← valB + valC;

   When 6 ⇒
   Case IF is

   When 0 ⇒ valE ← valB + valA;

   When 1 ⇒ valE ← valB − valA;

   When 2 ⇒ valE ← valB & valA;

   When 3 ⇒ valE ← valB ^ valA;

   When 7 ⇒
   Case IF is

   When 2 ⇒ If ZF then PC ← PC + valC;

   When 3 ⇒ If SF then PC ← PC + valC;

   When 8 ⇒
   valE ← valB − valA;

   When 9 ⇒ PC ← R[rsp];

5. **MEMORY ACCESS**: This step is performed if data is to be transferred between the CPU and external memory.

   If IC:IF = 41 M[valE]...M[valE+7] ← R[rA];

   if IC:IF = 42 R[rA] ← M[valE]...M[valE+7];

6. **WRITE BACK**: This step is performed if data is to be stored within the CPU.

   if IC:IF = 40 R[rB] ← valE;

   if IC = 6 R[rB] ← valE;

   if IC = 8 ZF ← (valE = 0), SF ← msb(valE);

7. **GO TO STEP 2**

The following example illustrates the application of the instruction execution algorithm on a simple machine language program that has been stored in memory:

The assembly language program and machine instructions :

instruction		semantics	byte1	byte2	byt3	byte4
mov	\$2, %r1	R[1] ← 2	60	01	02	00
mov	\$5, %r2	R[2] ← 5	60	02	05	00
add	%r1, %r2	R[2] ← R[2] + R[1]	40	12		

The machine language program loaded into memory at starting address 0xA00 :

address	contents
9FF	
A00	60
A01	01
A02	02
A03	00
A04	40
A05	02
A06	05
A07	00
A08	40
A09	12
A0A	

Applying the algorithm to the stored program :

Retrieving and Executing 1st Instruction :

1. **Step 1**: PC ← 0xA00
2. **Step 2**:
   IC:IF ← M[A00] = 40 (IC = 4, IF = 0)
   PC ← 0xA01
   IC = 4 ⇒
   (a) rA:rB ← M[A01] = 01
   (b) PC ← 0xA02
   (c) valC(7:0) ← M[A02] = 02
   (d) PC ← 0xA03
   (e) valC(15:8) ← M[A03] = 00
   (f) PC ← 0xA04
3. **Step 3**:
   valA ← R[0]
   valB ← R[1]
4. **Step 4**:
   IC = 4 ⇒ valE ← 0 + valC = 2
5. **Step 5**: not applicable

6. **Step 6**:
   IC:IF = 40 ⇒ R[1] ← valE = 2

Retrieving and Executing 2nd Instruction :

1. **Step 2**:
   IC:IF ← M[A04] = 40 (IC = 4, IF = 0)
   PC ← 0xA05
   IC = 4 ⇒
   (a) rA:rB ← M[A05] = 02
   (b) PC ← 0xA06
   (c) valC(7:0) ← M[A06] = 05
   (d) PC ← 0xA07
   (e) valC(15:8) ← M[A07] = 00
   (f) PC ← 0xA08
2. **Step 3**:
   valA ← R[0]
   valB ← R[2]
3. **Step 4**:
   IC = 4 ⇒ valE ← 0 + valC = 5
4. **Step 5**: not applicable
5. **Step 6**:
   IC:IF = 40 ⇒ R[1] ← valE = 5

Retrieving and Executing 3rd Instruction :

1. **Step 2**:
   IC:IF ← M[A08] = 60 (IC = 6, IF = 0)
   PC ← 0xA09
   IC = 6 ⇒
   (a) rA:rB ← M[A09] = 12
   (b) PC ← 0xA02A
2. **Step 3**:
   valA ← R[1]
   valB ← R[2]
3. **Step 4**:
   IC = 6 ⇒ IF= 0 ⇒ valE ← valB + valA = 5 + 2 = 7
4. **Step 5**: not applicable
5. **Step 6**:
   IC = 6 ⇒ R[2] ← valE = 7

29 CPU DATAPATH DESIGN

The datapath of the CPU provides all the components necessary to perform any arithmetic, logic, or data transfer tasks identified in the behavioural description; that is, in the instruction execution algorithm. For the current algorithm, these tasks can be identified by the register transfer statements within the algorithm.

For efficiency, it is convenient to try and implement each register transfer statement as a μ-instruction. A "μ-instruction" is a computation that can be performed by the datapath in a single assignment of values to the control word of the datapath.

To help identify what components will be required, it is useful to group these statements by function:

Summary of μ-Instructions

PC Update : These statements identify all the possible values that might be used to update the program counter register (PC).

1. PC ← <start address>
2. PC ← PC + 1
3. PC ← R[rsp]
4. PC ← PC + valC

Instruction Fetch : To complete the instruction fetch phase, the instruction retrieved from memory is stored in a set of registers, collectively called the "Instruction Register (IR)." Each of the variables IC, IF, rA, rB, and valC represent field of the IR and the location of these fields is determined by the physical instruction formats previously specified in the instruction set architecture.

1. IC:IF ← M[PC]
2. rA:rB ← M[PC]
3. valC(7:0) ← M[PC]
4. val(15:8) ← M[PC]

Arithmetic/Logic Computation : The arithmetic, logic, and shifting operations provided by the instruction set are used to determine the type of ALU that will be used and any additional computation hardware that may also be required.

valE ← valB op valA (op ∈ { +, −, &, ^ }

Condition Codes Update : Identifies the additional storage required for monitoring comparisons and other conditions

1. ZF ← (valE = 0)
2. SF ← msb(valE)

Register File Retrieval : Identifies the internal memory requirements for data retrieval from internal memory

1. valA ← R[rA]
2. valB ← R[rB]

Register File Update : Identifies the internal memory requirements for data storage within internal memory

1. R[rA] ← M[valE]. . .M[valE + 7]
2. R[rB] ← valE

Memory Update : Identifies the data transfer requirements for moving data from the CPU to external memory.

M[valE]. . .M[valE + 7] ← R[rA]

29.1 Instruction Fetch Hardware

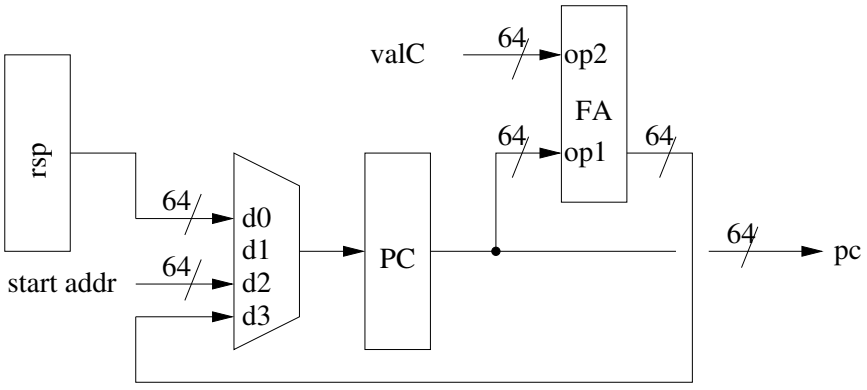
The following schematic is derived from the selection of components needed to meet the μ-instruction needs that were identified from the instruction execution algorithm. In particular:

Counter Register (PC) : to implement the Program Counter.

Storage Register (rsp) : to hold the STOP address.

4×64 Multiplexer : to choose among 3 possible values to load into the PC. (port d1 not used).

64-bit full adder : To compute effective addresses for base+displacement mode operands.



NOTE: Control inputs on all components are not shown.

29.2 Memory Hardware (Instruction Fetch)

To complete the instruction fetch phase requires the creation of an "instruction Register" to hold up to 4-bytes of an instruction retrieved from external memory. The value stored in the PC provides the address of the next byte of an instruction stored in external memory. As each byte is output it is stored in one of the four 8-byte registers that make up the 32-bit instruction register.

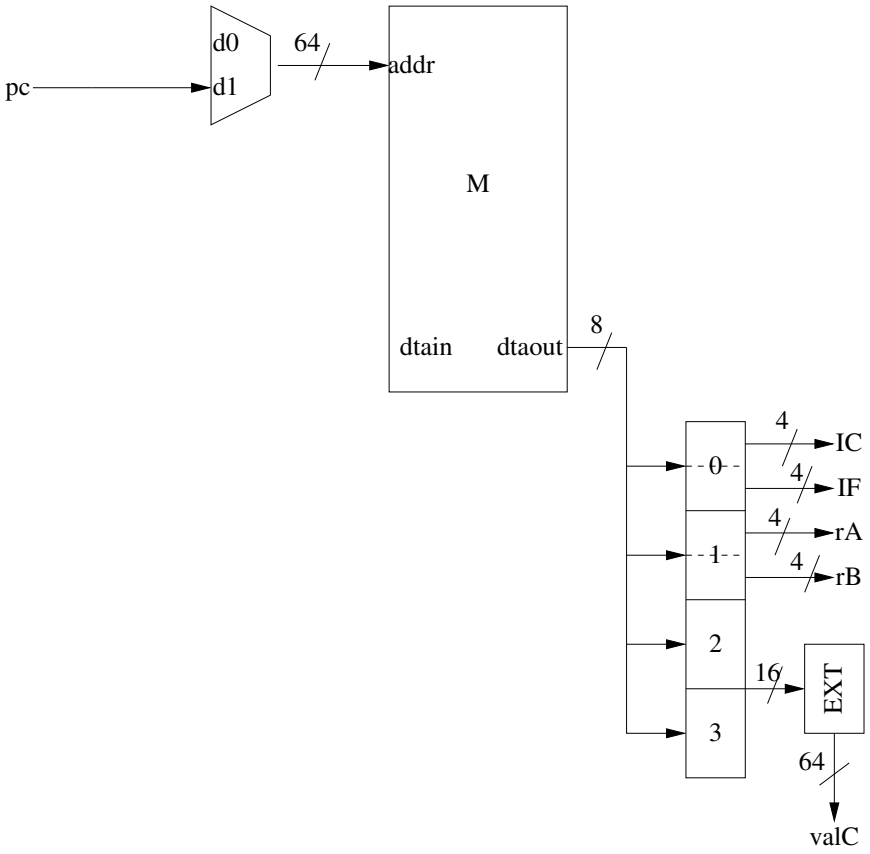
The components used are as follows:

2×64 Multiplexer : To select the address to be delivered to the address port of the external memory, M. Since both data and instructions are stored in memory but in different locations, there are two possible sources for addresses. One of these is from the PC and that address is used to specify the location of the first byte of an instruction.

$2^{64} \times 8$ **Random Access Memory (RAM)** : provides the storage for external memory. Note that this is not part of the CPU but is shown in the schematic to identify the interface between M and the CPU.

Four 8-bit Storage Registers : To hold up to four bytes of a retrieved instruction so that the opcode and operands can be extracted.

Sign Extender (EXT) : to convert a 16 bit value that is provided within the instruction format into a 64-bit value so that its size is compatible with the 64-bit architecture of the rest of the CPU.



NOTE: Control inputs on all components are not shown.