

|  |  |
| --- | --- |
|  |  |
| **Subject Name:** | Computer Organization and Architecture |
| **Subject Code:** | CSIS2083 |

**Report Title:** An Example of CPU Operation

**By**

|  |  |  |
| --- | --- | --- |
| **Student Name:** | Victor Tan Wei Kiat |  |
| **Student ID:** | D180138B |  |
| **Class ID:** | IT-18B |  |

|  |  |
| --- | --- |
| **Lecturer Name:** | Dr. Lee Huah |
| **Submit Date:** | 12th November 2019 |

**Table of Contents**

[**1** **The Fetch-Execute Cycle** 2](#_Toc24463798)

[**2** **Representing Programs** 3](#_Toc24463799)

[**3** **CPU\_VTWK Architecture** 4](#_Toc24463800)

[**4** **CPU\_VTWK Instruction Format** 5](#_Toc24463801)

[**5** **Memory Placement of Program and Data** 6](#_Toc24463802)

[**6** **CPU Organization** 7](#_Toc24463803)

[**6.1** **Interlude: The Von Neumann Machine Model** 8](#_Toc24463804)

[**6.2** **Instruction Execution (Fetch-Execute-Cycle Micro-steps)** 8](#_Toc24463805)

[**6.3** **LOAD R2, [201H]** 9](#_Toc24463806)

[6.3.1 Control Unit Action 9](#_Toc24463807)

[6.3.2 DECODE INSTRUCTION 9](#_Toc24463808)

[6.3.3 EXECUTE INSTRUCTION18 9](#_Toc24463809)

[**6.4** **ADD R2, [202H]** 10](#_Toc24463810)

[6.4.1 Control Unit Action 10](#_Toc24463811)

[6.4.2 DECODE INSTRUCTION 10](#_Toc24463812)

[6.4.3 EXECUTE INSTRUCTION18 10](#_Toc24463813)

[**6.5** **SUB R2, [203H]** 11](#_Toc24463814)

[6.5.1 Control Unit Action 11](#_Toc24463815)

[6.5.2 DECODE INSTRUCTION 11](#_Toc24463816)

[6.5.3 EXECUTE INSTRUCTION18 11](#_Toc24463817)

[**6.6** **DIV R2, [204H]** 12](#_Toc24463818)

[6.6.1 Control Unit Action 12](#_Toc24463819)

[6.6.2 DECODE INSTRUCTION 12](#_Toc24463820)

[6.6.3 EXECUTE INSTRUCTION18 12](#_Toc24463821)

[**6.7** **MULTI R2, [205H]** 13](#_Toc24463822)

[6.7.1 Control Unit Action 13](#_Toc24463823)

[6.7.2 DECODE INSTRUCTION 13](#_Toc24463824)

[6.7.3 EXECUTE INSTRUCTION18 13](#_Toc24463825)

[**6.8** **STORE R2, [200H]** 14](#_Toc24463826)

[6.8.1 Control Unit Action 14](#_Toc24463827)

[6.8.2 DECODE INSTRUCTION 14](#_Toc24463828)

[6.8.3 EXECUTE INSTRUCTION18 14](#_Toc24463829)

# **The Fetch-Execute Cycle**

The operation of the CPU[[1]](#footnote-1) is usually described in terms of the Fetch-Execute cycle[[2]](#footnote-2).

|  |  |
| --- | --- |
| **Fetch-Execute Cycle** | **The cycle raises many interesting questions, e.g.** |
| Fetch the *Instruction* | An instruction is referred to as a single action that will be performed by the CPU. The instruction is in the main memory. Instruction needs to be fetched to execute. It is not ok where it is because the storage memory will be filled up which causes the process to stop running. They keep track of instructions by using one of the register program contours to keep track of where it is in the program that is executing. Once the instruction is fetched, it needs to be decoded and executed. This could involve putting the instruction in the ALU then taking a different value from a register and adding the two together. |
| Increment the *Program Counter* | A Program Counter is a register in the control unit of the CPU that is used to hold the address of the current or next instruction. The Program Counter counts in binary. It also increases the stored value by 1 after each intrusion is fetched. The Program Counter is incremented after fetching an instruction and it holds the memory address of the next instruction that will be executed. |
| Decode the Instruction | The instruction is decoded because the process allows the CPU to know what instruction is needed to perform or executed so that the CPU knows how much operands it must fetch to perform the instruction. The opcode from the memory will then be decoded then being moved to the next appropriate registers. The encoded instruction represents as I in the Instruction Register (IR) is interpreted by the decoder. For the operation, it fetches a register and its sign extend an immediate value are added. The CPU must know when it sees the add instruction and gets two value from the register but add sign-extend in medicating value. |
| Fetch the *Operands* | In computer, an operand is the part of a computer instruction that specifies data that is to be operating on a manipulated and by extension, the data itself. It is the different fetching from step 1 above, the instruction describes operation and the operand on which the operation to be performed. |
| Perform the Operation | It is the main step of the whole fetch cycle and cannot be done simply. The operation is an elementary operation0 that a computer is designed and built to perform. |
| Store the results | The result generated by the operation will be moved from the register to memory or an output device. |
| Repeat forever | How do these steps execute any instructions at all?  Repeat the instruction operation, repeat for execute the new instruction operation and when the operation has error occurs, it might cause an infinite loop. |

In order to appreciate the operation of a computer, we need to answer such questions and to consider in more detail the organization of the CPU.

# **Representing Programs**

Each complex task carried out by a computer needs to be broken down into a sequence of simpler tasks and a **binary machine instruction** is needed for the most primitive tasks. Consider a task that adds two numbers[[3]](#footnote-3), held in memory locations designated by B, C, D, E, and F[[4]](#footnote-4) and stores the result in memory location designated by A.

A = [[(B +C) – D] / E] \*F

This assignment can be broken down (compiled) into a sequence of simpler tasks or **assembly instructions**, e.g.

|  |  |
| --- | --- |
| **Assembly Instruction** | **Effect** |

LOAD R2, B Copy the contents of memory location designated by B into Register 2

ADD R2, C Add the contents of the memory location designated by C to the contents of Register 2 and put the result back into Register

SUB R2, D Subtract the contents of the memory location designated by D to the contents of Register 2 and put the result back into Register 2

DIV R2, E Divide the contents of the memory location designated by E to the contents of Register 2 and put the result back into Register 2

MULTI R2, F Multiply the contents of the memory location designated by F to the contents of Register 2 and put the result back into Register 2

STORE R2, A Copy the contents of Register 2 into the memory location designated by A.

Each of these assembly instructions needs to be encoded into binary for execution by the Central Processing Unit (CPU). Let’s try this encoding for CPU\_VTWK, a simple architecture.

# **CPU\_VTWK Architecture**

CPU\_VTWK is a fictitious architecture with the following characteristics:

1024 x 16-bit words of RAM maximum. RAM is word-addressable.

4 general purpose registers R0, R1, R2 and R3. Each general purpose register is 16bits (the same size as a memory location).

16 different instructions that the CPU can decode and execute, e.g. LOAD, STORE, ADD, SUB, MULTI, DIV and so on. These different instructions constitute the **Instruction Set** of the Architecture.

The representation for integers will be two’s complement.

For this architecture, the computer architect needs to define a coding scheme[[5]](#footnote-5) for instructions. This is termed the **Instruction Format**.

# **CPU\_VTWK Instruction Format**

CPU\_VTWK instructions are 16-bits, to fit into a main-memory word. Each instruction is divided into a number of **instruction fields** that encode a different piece of information for the CPU.

Field Name

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OPCOD**  4-bits | | | **E** | **REG**  2-bits | | **ADDRESS**  10-bits | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Field Width

The **OPCODE**6 field identifies the CPU operation. Since CPU\_VTWK supports 16 instructions, these can be encoded as a 4-bit natural number. For CPU\_VTWK, opcodes 1 to 4 will be[[6]](#footnote-6):

0001 = LOAD 0010 = STORE 0011 = ADD 0100 = SUB 0101=DIV 0111 = MULTI

The **REG** field defines a General CPU Register. Arithmetic operations will use 1 register **operand** and 1 main memory **operand**, results will be written back to the register. Since CPU\_VTWK has 4 registers; these can be encoded as a 2-bit natural number:

00 = Register 0 01 = Register 1 10 = Register 2 11 = Register 3

The **ADDRESS** field defines the address of a word in RAM. Since CPU\_VTWK can have up to 1024 memory locations; a memory address can be encoded as a 10-bit natural number. If we define addresses 200H, 201H, 202H, 203H, 204H and 205H for A, B and C, we can encode the example above as:

|  |  |
| --- | --- |
| **Assembly Instruction** | **Machine Instruction** |
|  | Opcode Reg Operand |
| LOAD R2, [201H] | 0001 10 10 0000 0001 |
| ADD R2, [202H] | 0011 10 10 0000 0010 |
| SUB R2, [203H] | 0100 10 10 0000 0011 |
| DIV R2, [204H] | 0101 10 10 0000 0100 |
| MULTI R2, [205H] | 0111 10 10 0000 0101 |
| STORE R2, [200H] | 0010 10 10 0000 0000 |

# **Memory Placement of Program and Data**

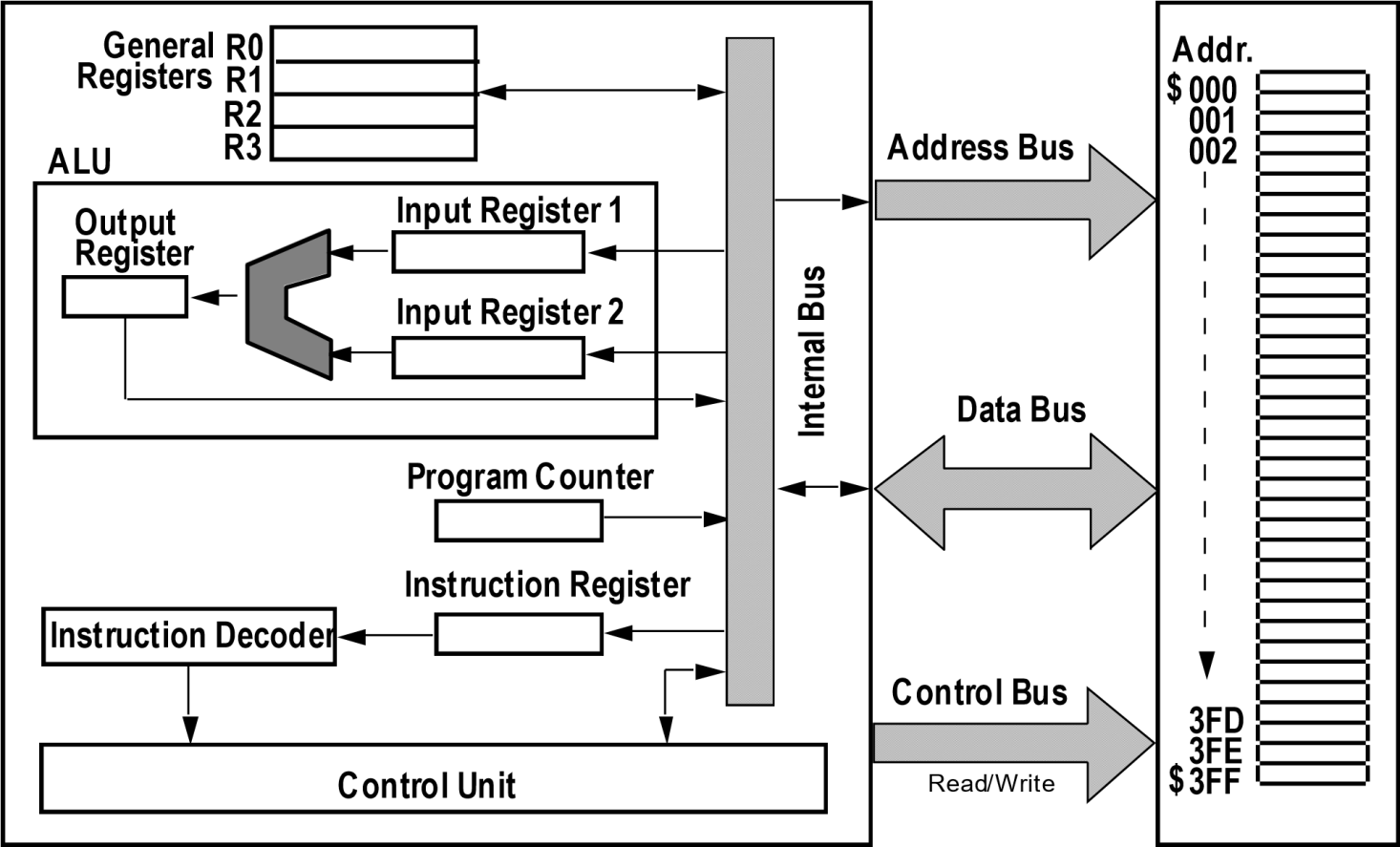
In order to execute a CPU\_VTWK program, its instructions and data needs to placed within main memory[[7]](#footnote-7). We’ll place our 3-instruction program in memory starting at address 080H and we’ll place the variables A, B and C at memory words 200H, 201H, and 202H respectively. Such placement results in the following memory layout prior to program execution. For convenience, memory addresses and memory contents are also given in hex.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Memory Address** in binary & hex | | | **Machine Instruction**  OP Reg Address | | | **Assembly Instruction** |
| 0000  0 | 1000  8 | 0000 0 | 0001  1 | 10 10  A | 0000 0001  0 1 | LOAD R2, [201H] |
| 0000  0 | 1000  8 | 0001 1 | 0011  3 | 10 10  A | 0000 0010  0 2 | ADD R2, [202H] |
| 0000  0 | 1000  8 | 0010 2 | 0100  4 | 10 10  A | 0000 0011  0 3 | SUB R2, [203H] |
| 0000  0 | 1000  8 | 0011 3 | 0101  5 | 10 10  A | 0000 0100  0 4 | DIV R2, [204H] |
| 0000  0 | 1000  8 | 0100 4 | 0111  6 | 10 10  A | 0000 0101  0 5 | MULTI R2, [205H] |
| 0000  0 | 1000  8 | 0101 5 | 0010  2 | 10 10  A | 0000 0000  0 0 | STORE R2, [200H] |
|  | **Etc** |  |  | **Etc** | | **Etc** |
| 0010  2 | 0000 0000  0 0 | | 0000 0000 0000 0000  0 0 0 0 | | | A = 0 |
| 0010  2 | 0000 0001  0 1 | | 0000 0000 0000 1000  0 0 0 8 | | | B = 8 |
| 0010  2 | 0000 0010  0 2 | | 0000 0000 0000 0110  0 0 0 6 | | | C = 6 |
| 0010  2 | 0000 0011  0 3 | | 0000 0000 0000 0010  0 0 0 2 | | | D = 2 |
| 0010  2 | 0000 0100  0 4 | | 0000 0000 0000 0011  0 0 0 4 | | | E = 4 |
| 0010  2 | 0000 0101  0 5 | | 0000 0000 0000 0110  0 0 0 3 | | | F = 3 |

Of course, the big question is “How is such a program executed by the CPU\_VTWK CPU?”

# **CPU Organization**

**Central Processing Unit (CPU) Memory**



The **Program Counter (PC)** is a special register that holds the **address** of the next instruction to be fetched from Memory (for CPU\_VTWK, the PC is 10-bits wide). The PC is incremented[[8]](#footnote-8) to "point to" the next instruction while an instruction is being fetched from main memory.

The **Instruction Register (IR)** is a special register that holds each instruction after it is fetched from main memory. For CPU\_VTWK, the IR is 16-bits since instructions are 16-bit wide.

The **Instruction Decoder** is a CPU component that decodes and interprets the contents of the Instruction Register, i.e. it splits a whole instruction into fields for the Control Unit to interpret. The Instruction decoder is often considered to be a part of the Control Unit.

The **Control Unit** is the CPU component that co-ordinates all activity within the CPU. It has connections to all parts of the CPU and includes a sophisticated timing circuit.

The **Arithmetic & Logic Unit (ALU)** is the CPU component that carries out arithmetic and logical operations e.g. addition, comparison, boolean AND/OR/NOT.

The **ALU Input Registers 1 & 2** are special registers that hold the input operands for the ALU.

The **ALU Output Register** is a special register that holds the result of an ALU operation. On completion of an ALU operation, the result is copied from the ALU Output register to its final destination, e.g. to a CPU register, or main-memory, or to an I/O device.

The **General-Purpose Registers R0, R1, R2, R3** are available for the programmer to use in his/her programs. Typically, the programmer tries to maximize the use of these registers in order to speed program execution. For CPU\_VTWK, the general registers are the same size as memory locations, i.e. 16-bits.

The **Buses** serve as communication highways for passing information within the CPU (CPU internal bus) and between the CPU and the main memory (the **address bus**, the **data bus**, and the **control bus**). The address bus is used to send addresses from the CPU to the main memory; these addresses indicate the memory location the CPU wishes to read or write. Unlike the address bus, the data bus is bi-directional; for writing, the data bus is used to send a word from the CPU to main-memory; for reading, the data bus is used to send a word from main-memory to the CPU. For CPU\_VTWK, the Control bus[[9]](#footnote-9) is used to indicate whether the CPU wishes to read from a memory location or write to a memory location. For simplicity we’ve omitted two special registers, the **Memory Address Register (MAR)** and the **Memory Data Register (MDR).** These registers lie at the boundary of the CPU and Address bus and Data bus respectively and serve to buffer data to/from the buses.

Buses can normally transfer more than 1-bit at a time. For the CPU\_VTWK, the address bus is 10bits (the size of an address), the data bus is 16-bits (size of a memory location), and the control bus is 1-bit (to indicate a memory read operation or a memory write operation).

## **Interlude: The Von Neumann Machine Model**

Most computers conform to the von Neumann machine model, named after the Hungarian American mathematician John von Neumann (1903-57).

In von Neumann’s model, a computer has **3 subsystems** (i) a CPU, (ii) a main memory, and (iii) an I/O system. The main memory holds the program as well as data and the computer is allowed to manipulate its own program[[10]](#footnote-10). Instructions are executed **sequentially** (one at a time). A single path exists between the control until and main-memory, this leads to the so-called "**von Neumann bottleneck**" since memory fetches are the slowest part of an instruction they become the bottleneck in any computation.

## **Instruction Execution (Fetch-Execute-Cycle Micro-steps)**

In order to execute our 3-instruction program, the control unit has to issue and coordinate a series of micro-instructions. These micro-instructions form the fetch-execute cycle. For our example we will assume that the Program Counter register (PC) already holds the address of the first instruction, namely 080H.

## **LOAD R2, [201H]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 1000 0000 | 0001 | 10 10 | 0000 | 0001 | Copy the value in memory word |
| 0 8 0 | 1 | A | 0 | 1 | 201H into Register 2 |

### Control Unit Action

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FETCH INSTRUCTION12** | Memory Address | **Data flows** | Memory Address |  |
| PC to Address Bus13 | 080H | **→** | 080H | Address Bus |
| 0 to Control Bus14 | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory | 080H | **→** | 080H | Memory |
| Control Bus to Memory | 0 | **Read→** | 0 | Memory |
| Increment PC15 | 080H | **INC→** | 081H | PC becomes PC+116 |
| Memory [080H] to Data Bus | 1A01H | **→** | 1A01H | Data Bus |
| Data Bus to Instruction Register | 1A01H | **→** | 1A01H | Instruction Register |

### DECODE INSTRUCTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IR to Instruction Decoder | 1A01H | **→** | 1A01H | Instruction Decoder |
| Instruction Decoder to Control Unit17 | 1,2, 201H | **→** | 1, 2, 201H | Control Unit |

OPCODE, REG and ADDRESS

### EXECUTE INSTRUCTION18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Control Unit to Address Bus | 201H | **→** | 201H | Address Bus |
| 0 to Control Bus | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory | 201H | **→** | 201H | Memory |
| Control Bus to Memory | 0 | **→** | 0 | Memory |
| Memory [201H] to Data bus | 0008H | **→** | 0008H | Data Bus (Content) |
| Data Bus to Register 2 | 0008H | **→** | 0008H | Register 2 |

1. The micro-steps in the Fetch and Decode phases are common for all instructions.
2. This and the next 4 micro-steps initiate a fetch of the next instruction to be executed, which is to found at memory address 80H. In practice a Memory Address Register (MAR) acts as an intermediate buffer for the Address, similarly a Memory Data Register (MDR) buffers data to/from the data bus.
3. We will use 0 for a memory READ request, and 1 for a memory WRITE request.
4. For simplicity, we will assume that the PC is capable of performing the increment internally. If not, the Control Unit would have to transfer the contents of the PC to the ALU, get the ALU to perform the increment and send the results back to the PC. All this while we are waiting for the main-memory to return the word at address 80H.
5. Since CPU\_VTWK’s main-memory is word-addressed, and all instructions are 1 word. If main-memory was byte addressed we would need to add 2.
6. The Instruction decoder splits the instruction into the individual instruction fields OPCODE, REG and ADDRESS for interpretation by the Control Unit.
7. The micro-steps for the execute phase actually perform the operation.

## **ADD R2, [202H]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 1000 0001 | 0011 | 10 10 | 0000 | 0010 | Add[[11]](#footnote-11) the value in memory word |
| 0 8 1 | 3 | A | 0 | 2 | 202H to Register 2 |

### Control Unit Action

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FETCH INSTRUCTION12** | Memory Address | **Data flows** | Memory Address |  |
| PC to Address Bus13 | 081H | **→** | 081H | Address Bus |
| 0 to Control Bus14 | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory | 081H | **→** | 081H | Memory |
| Control Bus to Memory | 0 | **Read→** | 0 | Memory |
| Increment PC15 | 081H | **INC→** | 082H | PC becomes PC+116 |
| Memory [081H] to Data Bus | 3A02H | **→** | 3A02H | Data Bus |
| Data Bus to Instruction Register | 3A02H | **→** | 3A02H | Instruction Register |

### DECODE INSTRUCTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IR to Instruction Decoder | 3A02H | **→** | 3A02H | Instruction Decoder |
| Instruction Decoder to Control Unit17 | 3, 2, 202H | **→** | 3, 2, 202H | Control Unit |

OPCODE, REG and ADDRESS

### EXECUTE INSTRUCTION18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register 2 to ALU Input Reg 1 | 0008H | **→** | 0008H | ALU Input Reg 1 |
| Control Unit to Address Bus | 202H | **→** | 202H | Address Bus |
| 0 to Control Bus | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory Control | 202H | **→** | 202H | Memory |
| Bus to Memory | 0 | **Read→** | 0 | Memory |
| Memory [202H] to Data bus | 0006H | **→** | 0006H | Data Bus (Data 6) |
| Data Bus to ALU Input Reg 2 | 0006H | **→** | 0006H | ALU Input Reg 2 |
| Control Unit to ALU |  | **Add→** | 000EH | Output Register |
| ALU Output Reg to Register 2 | 000EH | **→** | 000EH | Register 2 (Data 14) |

## **SUB R2, [203H]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 1000 0010 | 0100 | 10 10 | 0000 | 0011 | Subtract[[12]](#footnote-12) the value in memory word |
| 0 8 2 | 4 | A | 0 | 3 | 203H to Register 2 |

### Control Unit Action

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FETCH INSTRUCTION12** | Memory Address | **Data flows** | Memory Address |  |
| PC to Address Bus13 | 082H | **→** | 082H | Address Bus |
| 0 to Control Bus14 | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory | 082H | **→** | 082H | Memory |
| Control Bus to Memory | 0 | **Read→** | 0 | Memory |
| Increment PC15 | 082H | **INC→** | 083H | PC becomes PC+116 |
| Memory [082H] to Data Bus | 4A03H | **→** | 4A03H | Data Bus |
| Data Bus to Instruction Register | 4A03H | **→** | 4A03H | Instruction Register |

### DECODE INSTRUCTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IR to Instruction Decoder | 4A03H | **→** | 4A03H | Instruction Decoder |
| Instruction Decoder to Control Unit17 | 4, 2, 203H | **→** | 4, 2, 203H | Control Unit |

OPCODE, REG and ADDRESS

### EXECUTE INSTRUCTION18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register 2 to ALU Input Reg 1 | 000EH | **→** | 000EH | ALU Input Reg 1 |
| Control Unit to Address Bus | 203H | **→** | 203H | Address Bus |
| 0 to Control Bus | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory Control | 203H | **→** | 203H | Memory |
| Bus to Memory | 0 | **Read→** | 0 | Memory |
| Memory [203H] to Data bus | 0002H | **→** | 0002H | Data Bus (Data 2) |
| Data Bus to ALU Input Reg 2 | 0002H | **→** | 0002H | ALU Input Reg 2 |
| Control Unit to ALU |  | **Subtract→** | 000CH | Output Register |
| ALU Output Reg to Register 2 | 000CH | **→** | 000CH | Register 2 (Data 12) |

## **DIV R2, [204H]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 1000 0011 | 0101 | 10 10 | 0000 | 0100 | Divide[[13]](#footnote-13) the value in memory word |
| 0 8 3 | 5 | A | 0 | 4 | 204H to Register 2 |

### Control Unit Action

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FETCH INSTRUCTION12** | Memory Address | **Data flows** | Memory Address |  |
| PC to Address Bus13 | 083H | **→** | 083H | Address Bus |
| 0 to Control Bus14 | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory | 083H | **→** | 083H | Memory |
| Control Bus to Memory | 0 | **Read→** | 0 | Memory |
| Increment PC15 | 083H | **INC→** | 084H | PC becomes PC+116 |
| Memory [083H] to Data Bus | 5A04H | **→** | 5A04H | Data Bus |
| Data Bus to Instruction Register | 5A04H | **→** | 5A04H | Instruction Register |

### DECODE INSTRUCTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IR to Instruction Decoder | 5A04H | **→** | 5A04H | Instruction Decoder |
| Instruction Decoder to Control Unit17 | 5, 2, 204H | **→** | 5, 2, 204H | Control Unit |

OPCODE, REG and ADDRESS

### EXECUTE INSTRUCTION18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register 2 to ALU Input Reg 1 | 000CH | **→** | 000CH | ALU Input Reg 1 |
| Control Unit to Address Bus | 204H | **→** | 204H | Address Bus |
| 0 to Control Bus | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory Control | 204H | **→** | 204H | Memory |
| Bus to Memory | 0 | **Read→** | 0 | Memory |
| Memory [204H] to Data bus | 0004H | **→** | 0004H | Data Bus (Data 4) |
| Data Bus to ALU Input Reg 2 | 0004H | **→** | 0004H | ALU Input Reg 2 |
| Control Unit to ALU |  | **Divide→** | 0003H | Output Register |
| ALU Output Reg to Register 2 | 0003H | **→** | 0003H | Register 2 (Data 3) |

## **MULTI R2, [205H]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 1000 0100 | 0110 | 10 10 | 0000 | 0101 | Multiply[[14]](#footnote-14) the value in memory word |
| 0 8 4 | 6 | A | 0 | 5 | 205H to Register 2 |

### Control Unit Action

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FETCH INSTRUCTION12** | Memory Address | **Data flows** | Memory Address |  |
| PC to Address Bus13 | 084H | **→** | 084H | Address Bus |
| 0 to Control Bus14 | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory | 084H | **→** | 084H | Memory |
| Control Bus to Memory | 0 | **Read→** | 0 | Memory |
| Increment PC15 | 084H | **INC→** | 085H | PC becomes PC+116 |
| Memory [084H] to Data Bus | 6A05H | **→** | 6A05H | Data Bus |
| Data Bus to Instruction Register | 6A05H | **→** | 6A05H | Instruction Register |

### DECODE INSTRUCTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IR to Instruction Decoder | 3A02H | **→** | 3A02H | Instruction Decoder |
| Instruction Decoder to Control Unit17 | 6, 2, 205H | **→** | 6, 2, 205H | Control Unit |

OPCODE, REG and ADDRESS

### EXECUTE INSTRUCTION18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register 2 to ALU Input Reg 1 | 0003H | **→** | 0003H | ALU Input Reg 1 |
| Control Unit to Address Bus | 205H | **→** | 205H | Address Bus |
| 0 to Control Bus | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory Control | 205H | **→** | 205H | Memory |
| Bus to Memory | 0 | **Read→** | 0 | Memory |
| Memory [205H] to Data bus | 0003H | **→** | 0003H | Data Bus (Data 3) |
| Data Bus to ALU Input Reg 2 | 0003H | **→** | 0003H | ALU Input Reg 2 |
| Control Unit to ALU |  | **Multiply→** | 0009H | Output Register |
| ALU Output Reg to Register 2 | 0009H | **→** | 0009H | Register 2 (Data 9) |

## **6.8 STORE R2, [200H]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000 1000 0101 | 0010 | 10 10 | 0000 | 0000 | Copy the value in Register 2 into |
| 0 8 5 | 2 | A | 0 | 0 | Memory word 200H |

### Control Unit Action

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FETCH INSTRUCTION12** | Memory Address | **Data flows** | Memory Address |  |
| PC to Address Bus13 | 085H | **→** | 085H | Address Bus |
| 0 to Control Bus14 | 0 | **→** | 0 | Control Bus |
| Address Bus to Memory | 085H | **→** | 085H | Memory |
| Control Bus to Memory | 0 | **Read→** | 0 | Memory |
| Increment PC15 | 085H | **INC→** | 086H | PC becomes PC+116 |
| Memory [085H] to Data Bus | 2A00H | **→** | 2A00H | Data Bus |
| Data Bus to Instruction Register | 2A00H | **→** | 2A00H | Instruction Register |

### 6.8.2 DECODE INSTRUCTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IR to Instruction Decoder | 2A00H | **→** | 2A00H | Instruction Decoder |
| Instruction Decoder to Control Unit17 | 2, 2, 200H | **→** | 2, 2, 200H | Control Unit |

OPCODE, REG and ADDRESS

### 6.8.3 EXECUTE INSTRUCTION18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Register 2 to Data Bus | 0009H | **→** | 0009H | Data Bus (Data 9) |
| Control Unit to Address Bus | 200H | **→** | 200H | Address Bus |
| 1 to Control Bus | 1 | **→** | 1 | Control Bus |
| Data Bus to Memory | 0009H | **→** | 0009H | Memory |
| Address Bus to Memory Control | 200H | **→** | 200H | Memory |
| Control Bus to Memory | 1 | **Write→** | 1 | Memory |

1. Central Processing Unit. [↑](#footnote-ref-1)
2. Sometimes called the Fetch-Decode-Execute Cycle. [↑](#footnote-ref-2)
3. Let’s assume they are held in two’s complement form. [↑](#footnote-ref-3)
4. A, B and C are actually main memory **addresses**, i.e. natural binary numbers. [↑](#footnote-ref-4)
5. Most architectures actually have different instruction formats for different categories of instruction. 6 Operation Code [↑](#footnote-ref-5)
6. The meaning of CPU operations is defined in the Architecture’s Instruction Set Manual. [↑](#footnote-ref-6)
7. The Operating System software is normally responsible for undertaking this task. [↑](#footnote-ref-7)
8. By the appropriate number of memory words. [↑](#footnote-ref-8)
9. Most control-buses are wider than a single bit, these extras bits are used to provide more sophisticated memory operations and I/O operations. [↑](#footnote-ref-9)
10. This type of manipulation is not regarded as a good technique for general assembly programming. [↑](#footnote-ref-10)
11. Using two’s complement arithmetic. [↑](#footnote-ref-11)
12. Using two’s complement arithmetic. [↑](#footnote-ref-12)
13. Using two’s complement arithmetic. [↑](#footnote-ref-13)
14. Using two’s complement arithmetic. [↑](#footnote-ref-14)