

Computer Architecture Project

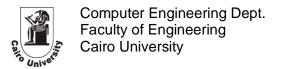
It is required to design simplified PDP11-based microprocessor that can execute the program loaded in its RAM.

Processor Specifications:

- Word Length: 16 bits
- Memory Size: 2K words
- Memory is word-addressable
- 4 Addressing modes
 - o Register mode "R0"
 - Auto-increment "(R0)+"
 - Auto-decrement "-(R0)"
 - Indexed "X(R0)"
 - The rest of the addressing modes are a Bonus:
 - Register mode indirect "@R0"
 - Auto-increment indirect "@(R0)+"
 - Auto-decrement indirect "@-(R0)"
 - Indexed indirect "@X(R0)"
- 8 General purpose registers (R0, R1, R2, R3, R4, R5, R6, R7)
 - R6 used as SP
 - o R7 used as PC
- 6 Special purpose registers (PC, SP, IR, MAR, MDR, FLAG)
 - o IR: Instruction Register
 - MAR: Memory Address Register
 - o MDR: Memory Data Register
 - FLAG: Status Flag Register contains (C, Z)
 - C: Carry Flag
 - Z: Zero Flag (1 if ALU result is 0)
 - N: Negative Flag (1 if ALU result is negative)

N.B:

You are free to add any number of internal registers that are hidden from the programmer's side.



Instruction Set:

Your design should support the following instruction set:

• 2 Operand Instructions

o Syntax "Opcode Src, Dst"

| Instruction | Operation Performed | |
|----------------------|--------------------------------------|--|
| MOV | Dst ← [Src] | |
| ADD | Dst ← [Dst] + [Src] | |
| ADC (Add with Carry) | Dst ← [Dst] + [Src] + C | |
| SUB | Dst ← [Dst] – [Src] | |
| SBC (Sub with Carry) | Dst ← [Dst] – [Src] – C | |
| AND | Dst ← [Dst] AND [Src] | |
| OR | Dst ← [Dst] OR [Src] | |
| XOR | Dst ← [Dst] XOR [Src] | |
| CMP (Compare) | [Dst] – [Src] | |
| | Neither of the operands are affected | |

• 1 Operand Instructions

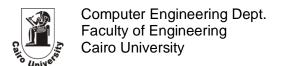
o Syntax "Opcode Dst"

| Instruction | Operation Performed |
|------------------------------|--|
| INC (Increment) | Dst ← [Dst] + 1 |
| DEC (Decrement) | Dst ← [Dst] – 1 |
| CLR (Clear) | Dst ← 0 |
| INV (Inverter) | Dst ← INV([Dst]) |
| LSR (Logic Shift Right) | Dst ← 0 [Dst] _{15->1} |
| ROR (Rotate Right) | Dst ← [Dst] ₀ [Dst] _{15->1} |
| ASR (Arithmetic Shift Right) | Dst ← [Dst] ₁₅ [Dst] _{15->1} |
| LSL (Logic Shift Left) | Dst ← [Dst] _{14->0} 0 |
| ROL (Rotate Left) | Dst ← [Dst] _{14->0} [Dst] ₁₅ |

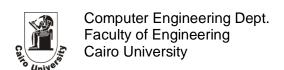
Branch Instructions

- o Syntax "Opcode Offset"
- o Operation "PC ← PC + Offset" is performed if branch condition is true

| Instruction | Branch Condition |
|-------------------------------|------------------|
| BR (Branch unconditionally) | None |
| BEQ (Branch if equal) | Z=1 |
| BNE (Branch if not equal) | Z=0 |
| BLO (Branch if Lower) | C=0 |
| BLS (Branch if Lower or same) | C=0 or Z=1 |
| BHI (Branch if Higher) | C=1 |



BHS (Branch if Higher or same) C=1 or Z=1



- No Operand Instructions
 - Syntax "Opcode"

| Instruction | Operation Performed |
|----------------------------|---------------------------|
| HLT (Halt) | Stop the processor |
| NOP (No Operation) | No operation is performed |
| | Continue code |
| RESET (Reset External bus) | All devices are reset |

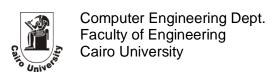
Jump Sub-Routine Instructions (Bonus)

| Instruction | Syntax | Operation Performed |
|------------------------------|-------------|-------------------------|
| JSR (Jump to subroutine) | JSR Address | SP ← [SP] – 1 |
| | | [SP] ← [PC] |
| | | [PC] ← [Address] |
| RTS (Return from subroutine) | RTS | [PC] ← [SP] |
| | | SP ← [SP] + 1 |
| | | |
| INTERRUPT | | save flags in stack |
| (Hardware Interrupt) | | save PC in stack |
| | | go to a certain address |
| IRET | IRET | pop flags from stack |
| | | return back to PC |

Phase1 Requirement:

In Phase 1, you should finish the design and analysis of your processor.

- 1] The design should have the details of the instruction set and processor architecture:
 - Instruction Set Architecture
 - IR structure for each instruction category
 - Bus System schematic including all connections with components:
 - o General Purpose Register File
 - Special Purpose Register File
 - o RAM
 - ALU with temp registers if needed (X / Y / Z)
 - o IR Decoding Unit
 - Bus/Busses structure and directions
 - Micro instructions
 - Assembler program
 - A small program to convert from Assembly syntax to Binary Opcode (any language)
 - Your assembler program should handle Immediate & Relative Addressing modes



2] You also should analyze your design in details:

- Number of Memory Access
 - Memory Access of each instruction
- Number of Clock cycles
 - Clock Cycles of each instruction
- Cycles per Instruction (CPI)
 - Average number of clock cycles per instruction
 "Sum(clock cycles of all instructions) / Number of instructions"

Delivery:

You will deliver a CD that contains the <u>assembler program</u> and a report (hardcopy or softcopy) that contains:

- 1. Team Number
- 2. Members Names
- 3. Bus System Schematic
- 4. Instruction Set Architecture
- 5. Micro-Instructions
- 6. Memory Access Analysis
- 7. Clock Cycles Analysis
- 8. CPI Analysis

Phase2 Requirement:

The implementation of your designed microprocessor using vhdl.

Delivery:

A CD containing:

- 1. Team Number
- 2. Members Names
- 3. All system VHDL files
- 4. All test assembly Memory files
- 5. Do Files / Testbench to test the system
- 6. Report that contains any design changes after phase 1

Groups:

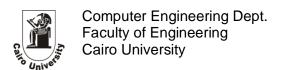
You will work in groups of 4 members.

Grading Criteria:

Phase 1 has 30% of the total project grade. Phase 2 has 70% of the total project grade.

Due Dates: These dates are subject to change by announcement

Phase1 delivery: 29 December 2020 Phase2 delivery: 19 January 2020

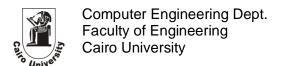


<u> Assembler Appendix:</u>

Your assembler should deal with the following:

- Variables
 - All variable will be at the end of the file with the following line format "Define Name Value". So in the test files "Define N 7" means that at this memory location there is a variable with value=7 and the variable name is N
 - o The variables are signed integers
- Immediate Addressing Mode
 - Immediate addressing mode has the following format "Opcode #0, R"
 - Your assembler should instead save the instruction as "Opcode (PC)+, R", and save in the next line "0".
- Relative Addressing Mode
 - Relative addressing mode has the following format "Opcode Var, R".
 - Your assembler should instead save the instruction as "Opcode X(PC), R", and save in the next line "VarAddress-LineNumber", where "VarAddress" is the variable address.
- JSR Addresses
 - JSR has the following format "JSR VarAddress"
 - Your assembler should instead save the instruction as "JSR Opcode",
 and save in the next line "VarAddress", where "VarAddress" is the sub-routine address.
- Offsets
 - Branch has the following format "Branch Offset"
 - Your assembler should instead save the instruction as "BR Opcode Offset"
 - The offset is a signed integer that is added directly to the updated PC
- Comments
 - o A semi colon starts a comment line

A sample test assembly code below.



Define M 5

;this code doesn't do anything significant, it is just an example

MOV N, R0 ; R0 = 7 address 0 XOR R1, R1 ; R1 = 0 address 2 MOV #20, R3 ; R3 = 20 address 3

; memory is word addressable, so there isno ; problem in having odd addresses, why?

address 19

Label3: ; address 5 MOV -(R3), M ; M = 5 , R3= 19 address 5

DEC R0 ; R0 = 6 address 7 CMP #18, @R3 ; C=1,N=1 address 8 BHI Label1 ; Not taken address 10 MOV #18, @R3 ; M=18 address 11

Label1: address 13 DEC_{R0} R0=5 address 13 BEQ Label2 address 14 not taken INC_{R3} R3=20 address 15 Label2: address 16 BR Label3 address 16 HLT address 17 Define N 7 address 18