Computer Architecture Project

Phase 1

Introduction:

It is required to design simplified PDP11-based microprocessor that can execute the program loaded in its ram. The microprocessor has the following characteristics:

- Word Length: 16 bits
- Memory Size: 4K words
- 32 Instruction
- 8 Addressing modes
  - Register mode “R0”
  - Auto-increment “(R0)+”
  - Auto-decrement “-(R0)”
  - Indexed “X(R0)”
  - Register mode indirect “@R0”
  - Auto-increment indirect “@(R0)+”
  - Auto-decrement indirect “@-(R0)”
  - Indexed indirect “@X(R0)”
- 8 General purpose register (R0, R1, R2, R3, R4, R5, R6, R7)
  - R7 used as PC
  - R6 used as SP
- 4 Special purpose register (IR, MAR, MDR, FLAG)
  - IR: Instruction Register
  - MAR: Memory Address Register
  - MDR: Memory Data Register
  - FLAG: Status Flag Register contains (C, Z, N, P, O)
    - C: Carry Flag
    - Z: Zero Flag (1 if ALU result is 0)
    - N: Negative Flag (1 if ALU result sign is Neg)
    - P: Parity Flag (1 if ALU result is even)
    - O: Overflow Flag (1 if P+P=N or N+N=P or P-N=N or N-P=P)

Instruction Set:

Your design should support the following instruction set:

- 2 Operand Instructions
  - 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]
XNOR	Dst ← [Dst] XNOR [Src]
CMP (Compare)	[Dst] – [Src] Neither of the operands are affected

- 1 Operand Instructions
  - 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 ← $[Dst]_{0}$ || $[Dst]_{15->1}$
RRC (Rotate Right with Carry)	Dst ← C || $[Dst]_{15->1}$
ASR (Arithmetic Shift Right)	Dst ← $[Dst]_{15}$ || $[Dst]_{15->1}$
LSL (Logic Shift Left)	Dst ← $[Dst]_{14->0}$ || 0
ROL (Rotate Left)	Dst ← $[Dst]_{14->0}$ || $[Dst]_{15}$
RLC (Rotate Left with Carry)	Dst ← $[Dst]_{14->0}$ || C

- Branch Instructions
  - Syntax “Opcode Offset”
  - 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

- Jump Sub-Routine Instructions (Bonus in Phase 2)

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 (Hardware Interrupt)	----------------	save flags in stack 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. The design should have the details of the instruction set and processor architecture:

- Instruction Set Architecture
  - Binary Opcode of each Instruction / mode
  - IR structure for each instruction category
- Bus System schematic including:
  - General Purpose Register File
  - Special Purpose Register File
  - RAM
  - ALU with temp registers if needed (temp1 / temp2 / temp3)
  - Bus/Busses structure and directions
- Micro-Instructions
  - Including Addressing Modes & Instruction Categories
- Assembler program
  - A small program to convert from Assembly syntax to Binary Opcode (any language)
  - Your assembler program should handle **Immediate** & **Absolute** Addressing modes

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 **assembler program** and a report (hardcopy and 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

Groups:

You will work in groups of 3~4 members.

Deadline & Discussion:

Delivery Deadline: Midnight Monday 2, December 2019

Discussion: Will be announced later.

Grading Criteria:

1. Bus System Schematic
    a. Completeness
    b. Understanding
2. Instruction Set Architecture
3. Micro-Instruction
4. Assembler Program
5. Performance Analysis
6. [Bonus] Best CPI using smallest number of busses

NB: Phase 1 has 30% of the total project grade.

Any Cheating case, both teams will be penalized as much as possible. So under any circumstances don’t cheat, just do your best and submit it before deadline.

Work should be almost equally divided between team mates of the same team, otherwise team mates of the same team will not get the same mark.

Any further modification to the document will be announced.

Assembler Appendix:

Your assembler should deal with the following:

- Variables
  - All variable will be at the end of the file with the following line format “#Value”.So in the test files “#7” means that at this memory location there is a variable with value=7
  - 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 (R2)+, R”, and save in the next line “0”.

- Absolute Addressing Mode
  - Absolute addressing mode has the following format “Opcode 0, R”.
  - Your assembler should instead save the instruction as “Opcode X(R2), R”, and save in the next line “0-LineNumber”, where “0” is the variable address.

- JSR Addresses
  - JSR has the following format “JSR R2, 0”
  - Your assembler should instead save the instruction as “JSR Opcode”, and save in the next line “0”, where “0” is the sub-routine address.

- Offsets
  - Branch has the following format “Branch 0”
  - Your assembler should instead save the instruction as “BR_Opcode Offset”, where Offset=0.
  - The offset is a signed integer

A set of sample test assembly codes is attached with the document.
NB: The test assembly file starts with line #0