Cairo University
Faculty of Engineering
Computer
Engineering Department

Third Year
Fall 2014

Architecture Project

Introduction:

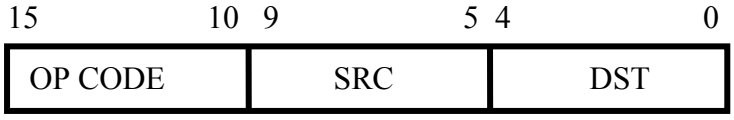
It is required to design and implement sythesizable simplified PDP11 microprocessor that can execute the program loaded in its ram. The PDP11 has the following characteristics:

- Word length 16 bits.
- Memory size of 2K words.
- Single bus architecture.
- 4 general purpose registers (R0, R1, R2, R3), (use R3 as PC, R2 as SP).
- 20 functions ALSU.
- Special purpose registers (IR, MAR, MDR, TMP, SRC, DST,FLAG_REGISTER).
- Support for the following 8 addressing modes:
    1. Register mode.
    2. Auto increment
    3. Auto decrement
    4. Indexed
    5. Register mode indirect
    6. Auto increment indirect
    7. Auto decrement indirect
    8. Indexed indirect.

The CPU support the following instruction set:

1. TWO WORD OPERAND:

Each of the SRC & DST are 5 bits, 2 for the register number and 3 for the addressing mode.



Mnemonic	Opcode	Operation preformed
Mov	**00 0000**	Dst ← [src]
Add	**00 0001**	Dst ← [dst] + [src]
Sub	**00 0010**	Dst ← [dst] - [src]
Bic (Bit clear)	**00 0100**	Dst←[dst] AND INV([src])
Bit Bit test	**00 1100**	[src] AND [dst] Neither of the operands are affected
Bis bit set (OR)	**00 0101**	Dst ← [dst] OR [src]
Xor	**00 0110**	Dst ← [dst] XOR [src]
Cmp (Compare)	**00 0010**	[src]-[dst] Neither of the operands are affected

2. ONE WORD OPERAND:

DST is 5 bits, 2 for the register number and 3 for the addressing mode

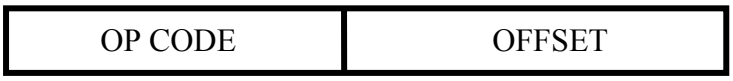
15 5 4 0

Mnemonic	Opcode(binary)	Operation preformed
Inc (Increment)	**10 00000 0000**	Dst ← [dst]+1
Dec (Decrement)	**10 00000 0011**	Dst ← [dst]-1
Clr (Clear)	**10 00001 0011**	Dst ← 0
Inv (invert)	**10 00000 0111**	Dst ← inv([src])
LSR (Logic shift right)	**10 00000 1000**	
ROR (Rotate right)	**10 00000 1001**	
RORC (Rotate right with carry)	**10 00000 1010**	
ASR (Arithmetic shift right)	**10 00000 1011**	
LSL (Logic shift left)	**10 00000 1100**	
ROL (Rotate left)	**10 00000 1101**	
ROLC (Rotate left with carry)	**10 00000 1110**	
ASL (Arithmetic shift left)	**10 00000 1111**	
JMP (unconditional jump)	**10 00010 0000**	Pc← Dst

3. BRANCH INSTRUCTIONS:

As our PDP11 is word addressable so we can't address one byte so the offset will be added as it is on the PC ie: (PC← PC +offset) and the offset is a 8 bit signed number.

15 8 7 0

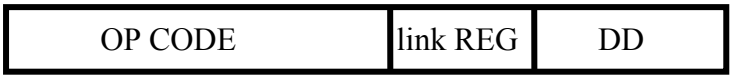


Mnemonic	name	Opcode (binary)	Branch condition
BR	Branch unconditionally	11 000 001	None
BNE	Branch if <>0	11 000 010	Z=0
BEQ	Branch if =0	11 000 011	Z=1

4. JUMP SUB ROUTINE INSTRUCTION: (BONUS)

DD are 5 bits, 2 for the register number and 3 for the addressing mode.

15 7 6 5 4 0



Mnemonic	name	Opcode (binary)	Operation preformed
JSR	(jump to subroutine)	11 001 0000	TEMP← DST SP←SP-1 [SP]←[REG] REG←[PC] PC←[TEMP]
RTS	(return from subroutine) the offset is don't care in this instruction	11 010 0000	PC←[REG] REG←[[SP]] SP←SP+1

5. NO OPERAND INSTRUCTIONS:

Mnemonic	name	Opcode (binary)	FUNCTION
HLT	Halt	11 000 000 00000000	Stops the processor
NOP	No operation	11 000 000 00000001	NO operation is preformed

Your design should include the following components:

1. **REGISTER ARRAY:** contains the registers R0, R1, R2, R3 connected to the bi-directional bus. You can use the one made in the lab but you will have to modify its control circuit (see page198 for details)
2. **SPECIAL REGISTERS ARRAY:** contains the PC, SP, TMP, SRC, DST registers connected to the bi-directional bus
3. **ALSU:** contains the ALSU with the Y and Z registers, beside the OVERFLOW, CARRY, ZERO, NEGATIVE flags.
   Note: find a suitable logic to force that the flags are only be affected by the main operation that the program executes (ex: when you increment the PC or when PCout, MARin, F=A the flags shouldn't change).
4. **DECODER SET:** combinational circuit consists of 7 or more decoders used to divide the control word (31 bits) into 11 functions f0 to f11 (book page 196).
5. **WIDE BRANCH:** your own combinational circuit to generate the address that would be ORED with the next address field from the ROM. This component is mainly responsible for generating the address of the wide branch after fetching the instruction from the memory to the IR, its inputs are IR &ROM[31..0]
6. **RAM:** contains the MAR, MDR and the RAM. Use the same module made in the lab. You will not need the WMFC because the RAM and ROM are on the same IC; but you can use it if you like. In both cases make sure that the memory output remains on the memory bus long enough so that the MDR can read the data.
   Note: as you have experienced in the ROM lab, you had to force the read or write signal in the next cycle after you load the address or data. From the architecture course point of view this is not correct because you write control signals like: PCout, MARin, RD on the same word of ROM.
   solutions: (YOU WILL BE ASKED ABOUT THIS PART)
   a) You can overcome this by introducing 2 registers after to maintain the RD & WR signals for another cycle (see book page 169).
   b) Insert and extra line below each RD and change the address of the book or make each RD microinstruction branch to another place in the ROM to waste a cycle then return back.
   c) Make MAR work on the –ve edge, and make the clock long enough for the memory to read then load to the MDR.
   d) Use the wmfc signal as an extention to the RD signal instead of buffering it.
7. **CPU:** contains the UAR, IR, ROM, and all the above modules.
   Note:
   1. The ALSU can be derived from the ROM (when executing microinstructions ex: INC PC…) or from the IR (when executing the operation in the main program after fetching its operands).
   2. The start computer signal is the clear signal to the PC and the UAR

3. The most important part of the processor is its ROM where you would spend most of the time writing its content, so you should frequently save it to different file versions, commenting its content is a must.
4. As we explained in the ROM lab there are 2 configurations concerning the clock.

    a. The control signals change on the –VE edge of the clock while all registers are +VE edge. This is easier to trace because the changes happen to registers in the same clock cycle but the disadvantages is that the maximum delay of your processor must be less than half clock cycle (the processor's clock must be doubled ,i.e.: slower processor).
    b. make both the processor and the microAR work on the +ve clock cycle (this is how the book is doing it).this is more difficult to trace because the changes to registers happen in the next clock cycle but this solution gives u the a complete clock cycle as a maximum propagation delay, i.e.: this is faster processor than the configuration above.

Building steps

1. You should integrate the above components (1, 2, 3, 6, and 7) into the CPU component.
2. Write the fetch words in the ROM then simulate to see that the instruction correctly moves to the IR. (WRITE ONLY THE FIRST 3 LINES IN THE ROM and check that they work)
3. From now on you will only change in the logic of the WIDEBRANCH beside continuing to fill up the ROM
4. Start taking each group of instructions and writing their control words in to ROM then simulate to insure that these instructions work correctly by writing a proper instruction in the RAM.

General advice:

- COMPILE YOUR DESIGN ON REGULAR BASES so that u can figure out what gave the new errors
- use the engineering sense to back trace the error source, that is remove the last thing u added, change the names of the wires to then see if the error is consistent.
- It is better to Draw the design before you implement it
- Whenever you get lost, you will find everything in you reference.
- **DON'T EVER IGNORE THE WARNINGS.**

Tips:

- You can enlarge the simulation time from file→end time
- Use suitable clock frequency then set the grid size to it, then apply the snap to grid option.
- If you stick to the word defined in the book you can copy the first 3 control words from the book to the ROM.
- Of course you will write it in VHDL
- Always save the ram and ROM files by export and import them.
- Make sure your code is synthesizable. This is a good presentation about how to make it so http://microe.udea.edu.co/~gpatino/digitalesii/Making_your_VHDL_synthesizable.pdf

Groups:

You should work in groups of 3-4.

Please fill the following sheet with your team members

https://docs.google.com/spreadsheets/d/1S-HqSLKTSSLMcTCuCwsViUXc-jAQWNRhLmQ8qE37qr0/edit#gid=0

How will the TA test your processor?

You will be given different memory initialization file (.mem) that contains different test programs. You are required to load it in the RAM, reset your processor to start and execute from memory location 0000. Each program would test some instructions (you should notify the TA if you haven't implemented or have logical errors concerning some of the instruction set), each file will include a small program will contain some instructions and some variables.
You MUST prepare a wave form editor with the main signals that show that the processor is working correctly (R0, R1, R2, R3), (PC, IR, MAR, MDR, TEMP, SRC, DST, Y, Z) (UAR,…).

Evaluation criteria:

As we have constructed most of the main components in the labs, 75 % of the mark will be on a working processor. This 75% will be divided upon the implementation and the understanding of the different instructions.

If you didn't integrate the previous components together or you have compilation errors in some modules, you will lose 75% of your mark and your degree will be based on the working components, in this case you should create a simulation file for each component and show the teacher assistant that every component is working properly.

Due date:

- MileStone1: 12 December at least one set on instruction done Midnight
- Final Due date is: 23 December 8am (after 4 weeks)
  Discussion will be on the same day isA

Deliverables

MileStone:

a compressed file with your teamNumber contains

1. your code ,
2. Your memory initalization files,
3. do files ,
4. 1-2 pages report about what is done and what is remaining
5. Schematic design of your processor
6. Your names should be written in the report

Final Delivery:

a compressed file with your teamNumber contains

1. your code ,
2. Your memory initalization files,
3. Sample programs
4. do files ,
5. Your names
6. yourselves