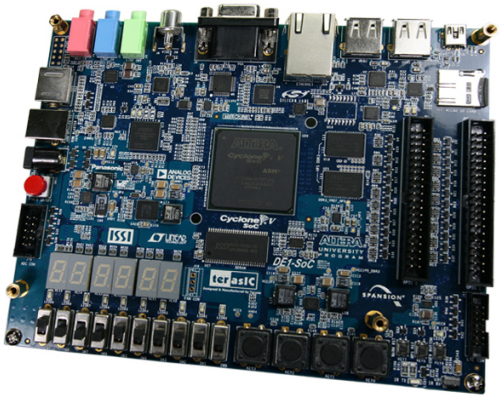
Digital System Design
Project - Computer and RPN Calculator

Design Manual and Instruction Set Architecture

Professor Jonathan H. Manton

A project for the subject:
Digital Systems (ELEN20006)



Electrical and Electronic Engineering
The University of Melbourne
Australia
2022

Contents

1 Interface **2**

2 Instruction Architecture **2**
2.1 Register File 2
2.1.1 Register 28 - Data Input Register (DINP) 3
2.1.2 Register 29 - General Output Register (GOUT) 3
2.1.3 Register 30 - Data Output Register (DOUT) 3
2.1.4 Register 31 - Flag Register (FLAG) 3
2.2 Program Memory 4
2.3 Instruction Set 4
2.3.1 Command Groups 4
2.3.2 Command 4
2.3.3 Arguments 1 and 2 4
2.3.4 Address 4

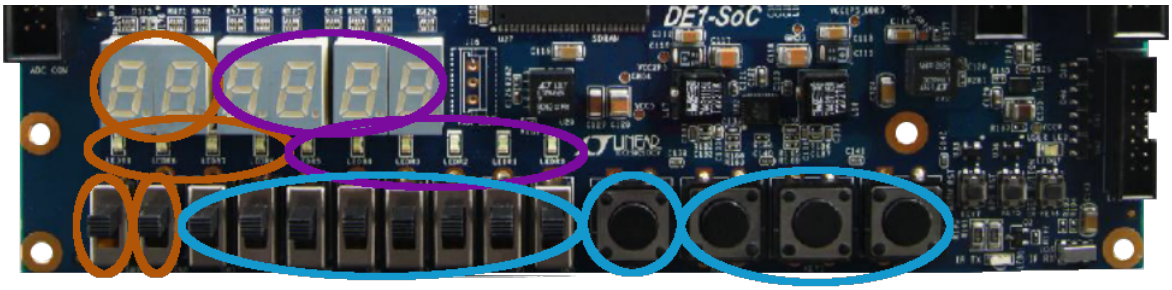
3 Command Groups **5**
3.1 No Operation (NOP) 5
3.2 Jump (JMP) 5
3.3 Atomic Test and Clear (ATC) 6
3.4 Move (MOV) 6
3.4.1 Shifting Operations 6
3.4.2 Shift Overflow 6
3.5 Accumulate (ACC) 7
3.5.1 Arithmetic Overflow 7
3.6 Remarks 7

4 CPU Design **7**
4.1 General Structure 8

5 System on Chip (SoC) Design **8**

1 Interface

A computer needs to have inputs and outputs to be useful.



Numbered from left to right, top to bottom:

1. Current contents of Instruction Pointer (IP) (hexadecimal)
2. Can be blank, or can display a decimal number between -128 and 127 inclusively
3. Available for hardware debugging
4. General Purpose Output: each LED can be turned on or off
5. Reset switch (HIGH to reset CPU)
6. Turbo switch (HIGH for turbo mode)
7. Data In: will be read by the CPU when the Sample Button is released
8. Sample Button (push then release for Data In to be read into CPU)
9. Buttons: CPU will be notified each time a push button is released

In the above list, items in orange are built into the computer hardware, items in purple are outputs under software control, and those in blue are inputs that can be read by the software. Note though that the software cannot read in the current values of the switches ("Data In") when it pleases. Rather, the CPU hardware arranges for the values of the switches to be latched into the CPU whenever the Sample button is released. This latched version is available for the software to read at any time.

2 Instruction Architecture

The architecture of our instruction set will be be very simple but not very efficient. The instructions will be 32 bits wide.

Our CPU will be able to access 256 32-bit memory locations (2048 bytes) and 32 8 bit registers.

Our CPU will implement 5 different command groups and each group can have a maximum of 8 commands.

2.1 Register File

The CPU will have access to 32 8-bit wide registers, numbered 0 to 31. Most of them are general purpose registers. However, the last four registers have special purposes. The registers are:

x0
x1
x2
...
x26
x27
DINP
GOUT
DOUT
FLAG

2.1.1 Register 28 - Data Input Register (DINP)

DINP holds the latched contents of the din CPU pins.

2.1.2 Register 29 - General Output Register (GOUT)

The contents of **GOUT** will appear on the reg_gout pins of the CPU.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data Valid	-	GPO[5]	GPO[4]	GPO[3]	GPO[2]	GPO[1]	GPO[0]
DVAL							

DVAL is used to signify that the data in the Data Output Register (DOUT) is valid.

2.1.3 Register 30 - Data Output Register (DOUT)

The contents of Register 30 (DOUT) will appear on the reg_dout output pins of the CPU.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Dout[7]	Dout[6]	Dout[5]	Dout[4]	Dout[3]	Dout[2]	Dout[1]	Dout[0]

2.1.4 Register 31 - Flag Register (FLAG)

Certain events will cause **FLAG** to be modified. The contents of **FLAG** will appear on the reg_flag pins of the CPU.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	Shift	Overflow	Sample	Btn 2	Btn 1	Btn 0
		SHFT	**OFLW**	**SMPL**			

If a push-button is pressed and then released, the corresponding bit of FLAG will be set to a 1. It will remain a 1 until cleared by the programmer. The Overflow bit will be set or cleared after each arithmetic operation that has the possibility of overflowing, such as addition or multiplication. The Shift bit will be modified after a shift operation has taken place; it is the value of the bit 'shifted out'.

2.2 Program Memory

The CPU will read instructions from the Program Memory, which is a block of memory containing 256 memory cells, each cell being 32 bits wide. This memory is read only and asynchronous.

The limitation here is that the address field of the instruction is only 8 bits wide. This means that the CPU can only address 256 memory cells.

2.3 Instruction Set

Each instruction is 32 bits wide: such a large width wastes memory but makes the job of both the programmer and designer easier. The programmer can write in machine code more easily due to the simple structure, and similarly, when we come to designing the CPU, it will be easier to decode each instruction. A 32 bit instruction is decomposed as follows.

Bits 31-29	Bits 28-26	Bits 25-17	Bits 16-8	Bits 7-0
Command Group	Command	Argument 1	Argument 2	Address

2.3.1 Command Groups

The first 3 bits of an instruction encode the Command Group.

3'b000	NOP	No Operations
3'b001	JMP	Jump
3'b010	ATC	Atomic Test and Clear
3'b011	MOV	Move
3'b100	ACC	Accumulate

2.3.2 Command

The next 3 bits of an instruction specify the Command. Their meaning depends on the Command Group. See Section 3 for details.

2.3.3 Arguments 1 and 2

The next 9 bits of an instruction comprise Argument 1 and the 9 bits after those comprise Argument 2. Arguments describe either an 8-bit number or a register location in one of two ways. Decompose the argument as a,b where a is a 1-bit number and b an 8-bit number. If a=0 then the number is simply b. For example, 9'b0_0000_1000 represents the number 8. If a register location is required, it is an error to use a=0. If a=1 then the location is Register b, and if a number is required, then the number is whatever is in Register b.

Whether a location or a number is expected will be clear from the Command Group.

2.3.4 Address

Finally, the remaining 8 bits of an Instruction store an Address that is used for the JMP and ATC Command Groups.

3 Command Groups

3.1 No Operation (NOP)

The NOP command does nothing (in a sense). When a NOP command is issued to the CPU, the default behaviour of the CPU is to increment the Instruction Pointer (IP) by 1. The NOP command is used to fill gaps in the instruction memory or they can be used to delay the CPU (This is generally not how a sleep / delay function in C works!).

3.2 Jump (JMP)

The JMP command group is used to change the IP. A jump command uses every part of the instruction (except for unconditional jumps).

The different commands that the jump command group uses are as follows:

Command	Abbreviation	Description
3'b000	UNC	Unconditional Jump
3'b010	EQ	Jump On Equality
3'b100	ULT	Jump On Unsigned Less Than
3'b101	SLT	Jump On Signed Less Than
3'b110	ULE	Jump On Unsigned Less Than Or Equal To
3'b111	SLE	Jump On Signed Less Than Or Equal To

For unconditional jumps, the instruction would look like this:

Bits 31-29	Bits 28-26	Bits 25-17	Bits 16-8	Bits 7-0
3'b001	3'b000	9'bxxxxxxxxx	9'bxxxxxxxxx	Address

When the CPU reads this instruction, the IP should be set to the address specified in the address field.

For the other commands, the CPU needs to make some comparison between argument 1 and argument 2. If the comparison is true, the IP should be set to the address specified in the address field.

For example, lets take an JMP ULT instruction. The instruction could look like this:

Bits 31-29	Bits 28-26	Bits 25-17	Bits 16-8	Bits 7-0
3'b001	3'b100	9'b0_0000_0001	9'b0_0000_0010	8'b0000_1000

The CPU would need to evaluate if 8'b0000_0001 is less than 8'b0000_0010. As it is, the IP should be set to 8'b0000_1000.

This could be read as:

```
if (1 < 2)
    goto 0x8; // Go to address 0x8 and start executing instructions there
```

3.3 Atomic Test and Clear (ATC)

The ATC Command Group is used to test a particular bit of FLAG. The 3-bit Command specifies which bit (from 0 to 7). If the bit is set, the IP should be set to the address specified in the address field. If the bit is not set, the IP should be incremented by 1.

An ATC instruction would look like this:

Bits 31-29	Bits 28-26	Bits 25-17	Bits 16-8	Bits 7-0
3'b010	Bit to check	9'bx_xxxx_xxxx	9'bx_xxxx_xxxx	Address

3.4 Move (MOV)

The MOV Command Group is used to move data from one location to another. The 3-bit Command specifies the type of move.

As it only makes sense to move numbers into a register, the MOV command group should always specify the second argument as a register.

The different commands that the MOV command group uses are as follows:

Command	Abbreviation	Description
3'b000	PUR	Pure Move
3'b001	SHL	Move and Shift Left
3'b010	SHR	Move and Shift Right

An MOV instruction would look like this:

Bits 31-29	Bits 28-26	Bits 25-17	Bit 16	Bits 15-8	Bits 7-0
3'b011	Move Type	Argument 1	1'b1	Register Number	8'dx

3.4.1 Shifting Operations

The shift commands shift the number in argument 1 left or right by one bit. For example, if argument 1 is 9'b0_0000_0001, then the SHL command would shift the number in argument 1 left by one bit and the result would be 8'b0000_0010.

3.4.2 Shift Overflow

If the resulting number that is being shifted is larger than 8 bits, the result is truncated to 8 bits. For example, if argument 1 is 9'b0_1000_0001, then the SHL command would shift the number in argument 1 left by one bit and the result would be 8'b0000_0010. This should produce a shift overflow flag as a bit has been lost. Therefore when this situation happens FLAG should have it's shift overflow bit set.

3.5 Accumulate (ACC)

The ACC command group is used to perform arithmatic operations and store the result in a register.

Again, for the command group to make sense, the second argument should always be a register (the result needs to be stored in a register).

The different commands that the accumulate command group uses are as follows:

Command	Abbreviation	Description
3'b000	UAD	Unsigned Addition
3'b001	SAD	Signed Addition
3'b010	UMT	Unsigned Multiplication
3'b011	SMT	Signed Multiplication
3'b100	AND	Bitwise AND
3'b101	OR	Bitwise OR
3'b110	XOR	Bitwise XOR

3.5.1 Arithmetic Overflow

Both addition and multiplication commands can cause integer overflow. If you have never heard of integer overflow, the basic premise is that when performing arithmatic operations on two integers, the result may be too big to store in the number of bits available. If this occurs in our CPU, like the shift overflow, the overflow bit should be set in the FLAG register.

3.6 Remarks

- There are many instructions CPUs normally have that we have omitted for simplicity. Nevertheless, this CPU already has more than enough instructions with which to implement an RPN calculator. You can always add more if you wish though!
- You are not expected how to implement the CPU just from reading this document. Although, at the end of this project, there is nothing stopping you from reading an open source ISA and implementing it yourself (see RISC V or MIPS).
- The instructions could be used more efficiently. Consider the fact that during an ACC command, the address bits aren't used. Different architectures may use those bits for something else. Perhaps a third register?

4 CPU Design

The CPU will be a single cycle processor, meaning all instrutions are executed in one cycle. This makes the CPU very simple to implement, at the cost of being very slow. This also means that reads from the instruction memory and registers need to be asynchronous.

The CPU will have the following I/O ports:

Name	Width	I/O	Description
clk	1	I	CPU clock
enable	1	I	CPU clock enable
resetn	1	I	CPU reset
instruction	32	I	Current instruction to execute
instruction_pointer	8	O	Pointer to current instruction
din	8	I	Latched data input - Connected to DIN in register file
gpi	4	I	Edge detected general purpose inputs - Connected to FLAG bits 3 to 0
reg_dout	8	O	Data output - Connected to DOUT in register file
reg_gout	8	O	Connected to GOUT in register file
reg_flag	8	O	Connected to FLAG in register file

4.1 General Structure

The CPU will mainly be split up into mainly 3 modules:

- The Control Unit: This module will contain the control logic for the CPU. It will control things like alu operations, register write enables and instruction pointer branch selection.
- The Arithmatic Logic Unit: This module will perform different mathematical operations that our different command groups require.
- The Register File: This module will contain the registers that the CPU will use.

There will also be an instruction pointer in the CPU. The basic idea is that the instruction pointer will increment by 1 each clock cycle, or be set to an address depending on the instructions given and current state of the CPU.

5 System on Chip (SoC) Design

Our project will include an SoC module. This module is responsible for connecting the CPU, instruction memory and the external interface together. It will also contain some logic to handle the external interface.

The SoC module will have the following I/O ports:

Name	Width	I/O	Description
clk	1	I	System clock
turbo_mode	1	I	Activates the 'fast' mode of the CPU
resetn	1	I	System reset
gpi	4	I	General purpose input - This will connect to the buttons
gpo	6	O	General purpose output - This will connect the the right 6 LEDs
din	8	I	Raw data input - Connected to the right 8 switches
dval	1	I	Data valid - Signifies to external interface that dout is valid
dout	8	O	Data output - Output data from CPU
debug	8	O	Debug signals - Useful for debugging signals using the left 4 LEDs
ip	8	O	Instruction pointer - Connected to the CPU instruction pointer

The SoC will contain the following modules:

- The CPU.
- The instruction memory.
- Modules related to conditioning the inputs (synchronisers, debouncers, falling edge detectors).
- The Enable Generator: A module to slow the clock enable line of the CPU.