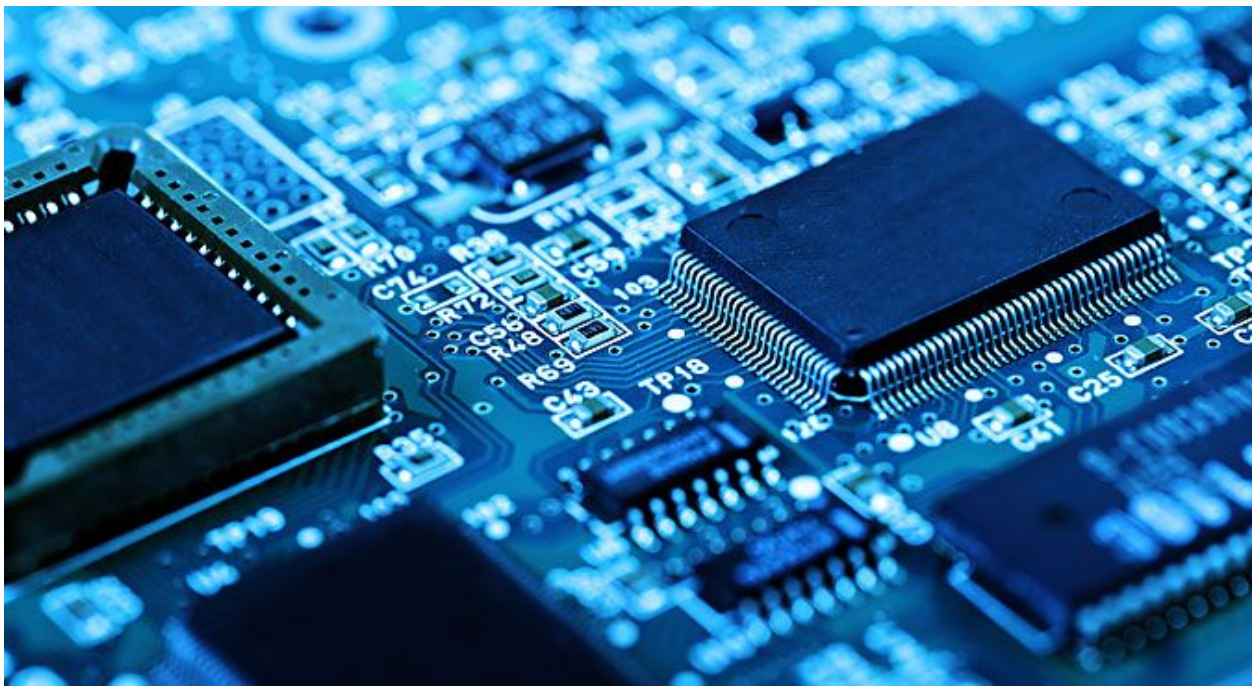


MICROPROCESSOR

Mini Project

VIGNESH S : EE16B127
SIDDHARTH VADAPALLI :EE16B125
08/11/2017

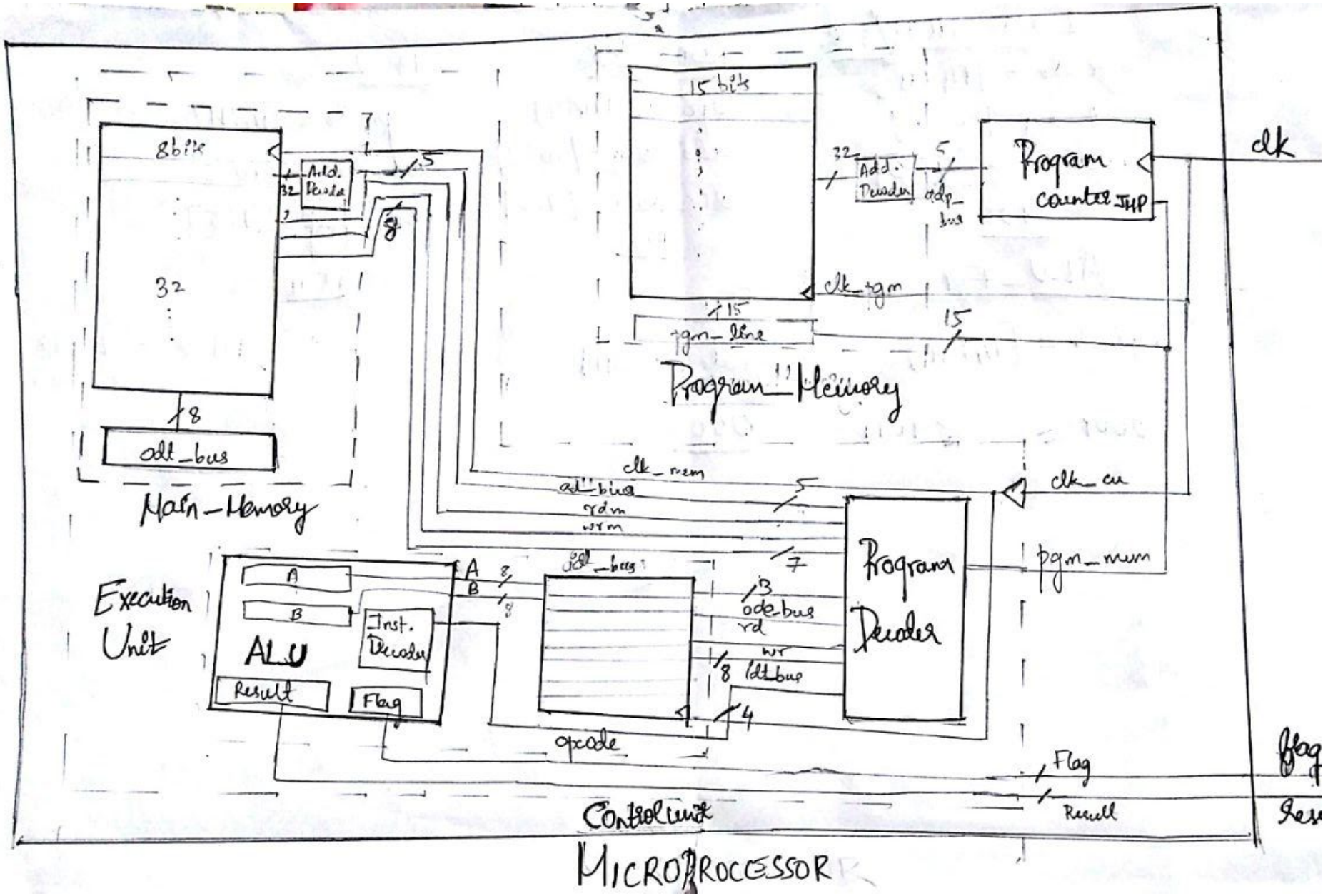
About the processor

The processor that has been designed here is a 8 bit Microprocessor. I have designed it using iverilog and have done the simulations using a software called GTKwave . The architecture of it is as follows :

- An ALU with 2 input registers and one register for the result (8-bit) and another flag register (2-bit).
- Six 8-bit internal registers .
- An Execution Unit (EU) which contains above two things.
- A Data Memory (DM) which has thirty two (32) 8- bit registers to store excess information.
- The Control Unit (CU) contains both EU and DM.
- A 15-bit (fifteen bit) program memory (PM) containing thirty two registers is present to write the program that has to be executed by the processor
- A 15-bit program counter (PC).
- The top most module is the processor which combines the CU, PM and PC.

BLOCK DIAGRAM

The picture below illustrates the architecture of the processor mentioned above along with the inputs and outputs going into and out of each module.


15 bits
8bits
32
Add. Decoder
32
5
Program Counter JMP
clk
clk_pgm
15
pgm_line
Program Memory
8
adr_bus
Main Memory
clk_mem
adr_bus
rdm
wrm
clk_cu
pgm_mem
Program Decoder
Execution Unit
A
B
ALU
Inst. Decoder
Result
Flag
A
B
8
8
adr_bus
7
3
ode_bus
rd
wr
8
Idt_bus
4
opcode
Flag
Result
Control Unit
MICROPROCESSOR

.

Functions performed by the processor

- It can ADD(0001), SUBTRACT(0010), AND(0011), OR(0100), LEFT SHIFT(0101), RIGHT SHIFT(0110), XOR(0111), COMPLEMENT(1000), COMPARE(1001) and MULTIPLY(1010). These are performed in the ALU.
- It can LOAD DIR(1110) directly a value.
- It can LOAD MEM(1100) from memory.
- It can STR(1101) address into memory.
- It can MOV (1011) value from register to another.
- It can JUMP(0000) from instruction to another.

INSTRUCTION sequence is as below:

[14:11] - opcode

[10:8] - address of register

[7:3] - address of memory

[7:0] - data value for load direct

[7:3] - instruction to jump

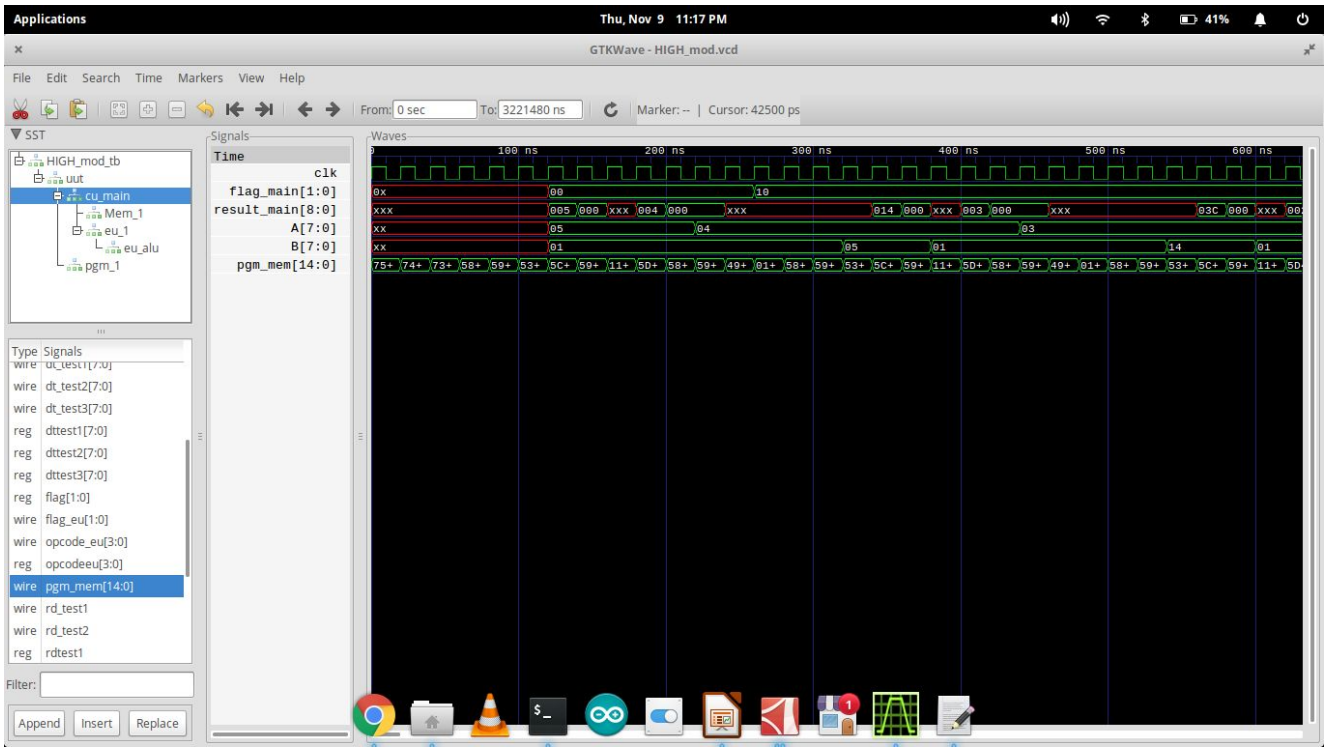
[10:8] - address of destination register

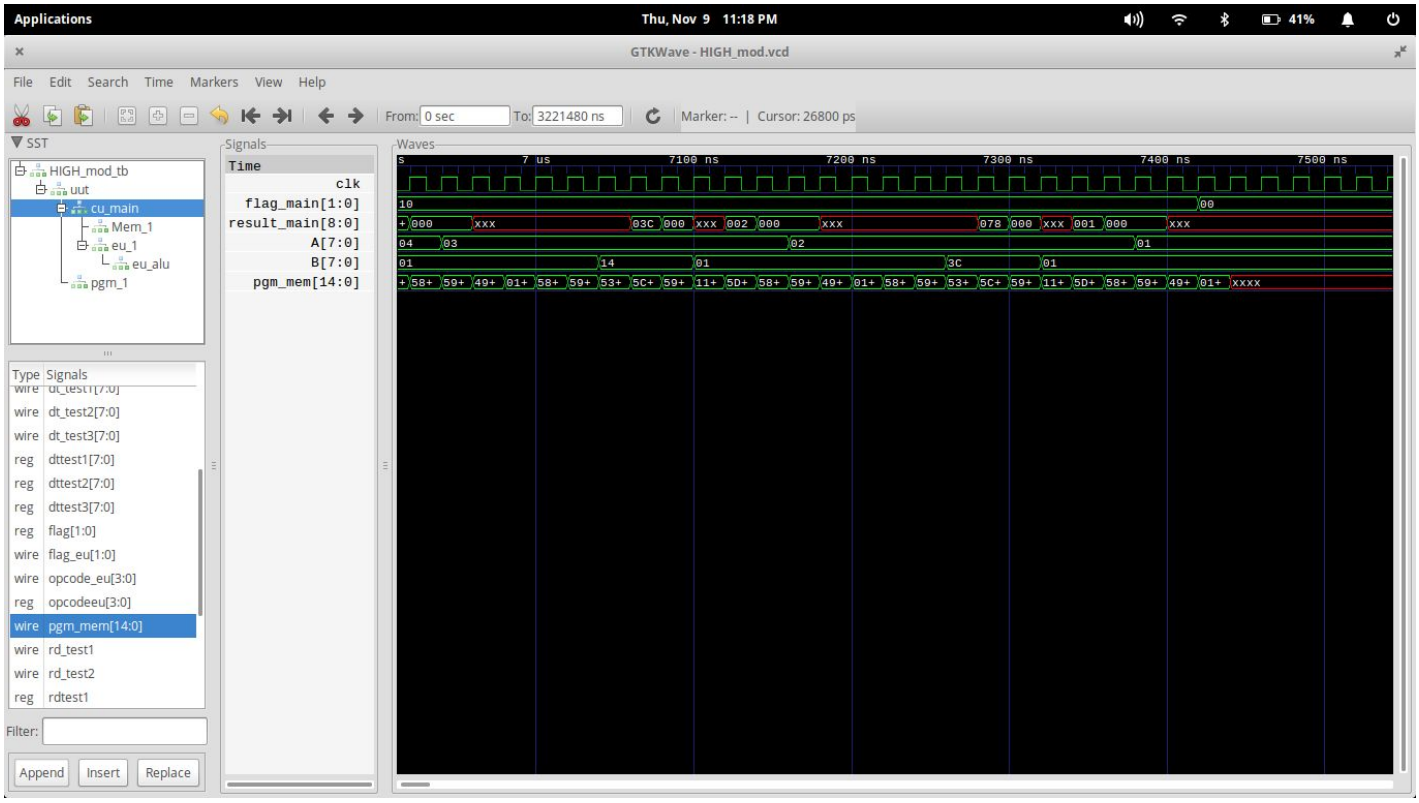
[7:5] - address of source register

Hard Coded Program for factorial

```
pgm_memory[0] = 15'b111000000000001;  //Factorial
code
pgm_memory[1] = 15'b111010100000101;  //LD data to R5
pgm_memory[2] = 15'b111010000000001;  //LD data to R4
pgm_memory[3] = 15'b111001100000001;  //LD data to R3
pgm_memory[4] = 15'b101100010100000;  //MOV R0,R5
pgm_memory[5] = 15'b101100110000000; //MOV R1,R4
pgm_memory[6] = 15'b101001100000001; //MUL RO,R1
pgm_memory[7] = 15'b101110001000000;  //MOV R4,R2
pgm_memory[8] = 15'b101100101100000;  //MOV R1,R3
pgm_memory[9] = 15'b001000101100000;  //SUB R0,R1
pgm_memory[10] = 15'b101110101000000;  //MOV R5,R2
pgm_memory[11] = 15'b101100010100000;  // MOV R0,R5
pgm_memory[12] = 15'b101100101100000;  //MOV R1,R3
pgm_memory[13] = 15'b100100110000000;  //CMP R0,R1
pgm_memory[14] = 15'b000000100100000;  //JMP
```

The above code basically finds the factorial of 5 using JUMP statements and BRANCH statements. The output of the above program is uploaded and answer is 078 in hexadecimal which is 120 in decimal which is factorial of 5.


Applications
Thu, Nov 9 11:17 PM
41%
GTKWave - HIGH_mod.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 3221480 ns
Marker: -- | Cursor: 42500 ps
SST
HIGH_mod_tb
uut
cu_main
Mem_1
eu_1
eu_alu
pgm_1
Signals
Time
clk
flag_main[1:0]
result_main[8:0]
A[7:0]
B[7:0]
pgm_mem[14:0]
Waves
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
Type Signals
wire dt_test2[7:0]
wire dt_test3[7:0]
reg dttest1[7:0]
reg dttest2[7:0]
reg dttest3[7:0]
reg flag[1:0]
wire flag_eu[1:0]
wire opcode_eu[3:0]
reg opcodeeu[3:0]
wire pgm_mem[14:0]
wire rd_test1
wire rd_test2
reg rdtest1
Filter:
Append
Insert
Replace


Applications
Thu, Nov 9 11:18 PM
41%
GTKWave - HIGH_mod.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 3221480 ns
Marker: -- | Cursor: 26800 ps
SST
HIGH_mod_tb
uut
cu_main
Mem_1
eu_1
eu_alu
pgm_1
Signals
Time
clk
flag_main[1:0]
result_main[8:0]
A[7:0]
B[7:0]
pgm_mem[14:0]
Waves
7 us
7100 ns
7200 ns
7300 ns
7400 ns
7500 ns
Type Signals
wire dt_test2[7:0]
wire dt_test3[7:0]
reg dttest1[7:0]
reg dttest2[7:0]
reg dttest3[7:0]
reg flag[1:0]
wire flag_eu[1:0]
wire opcode_eu[3:0]
reg opcodeeu[3:0]
wire pgm_mem[14:0]
wire rd_test1
wire rd_test2
reg rdtest1
Filter:
Append
Insert
Replace