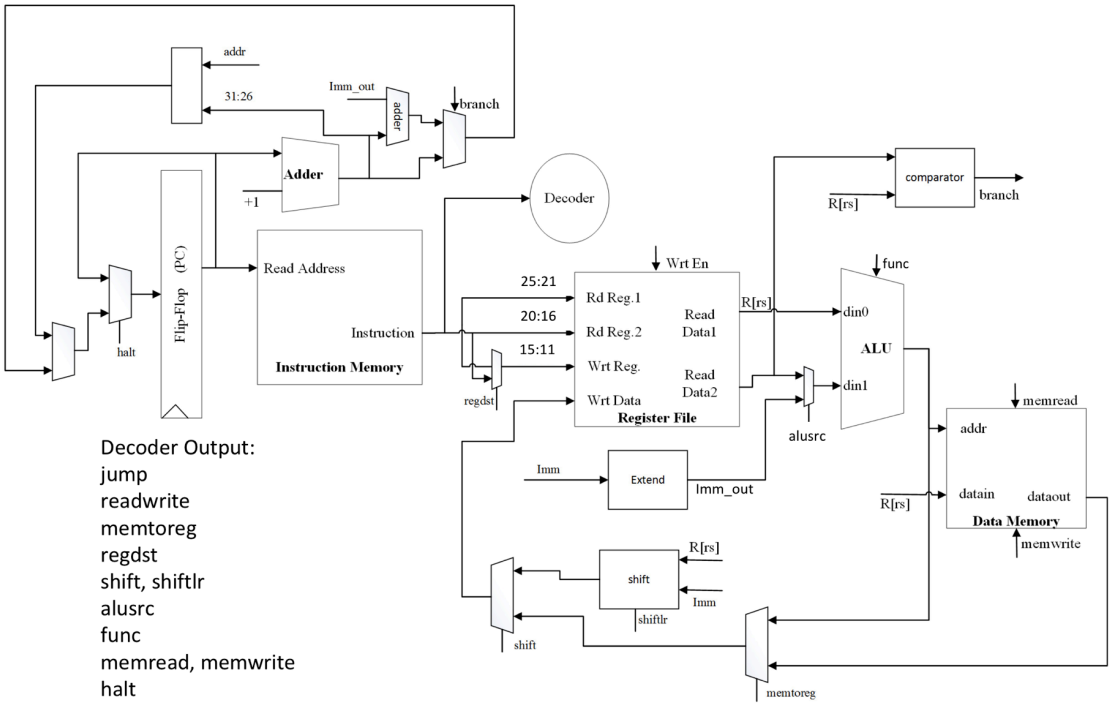
EL6463
ADV Hardware Computer Design

Xuetao Hong
Junlin Liu
Shao Yue
Yufei Wang
Bolei Zhang
Yuqi Zhang

Final Project Report
MIPS Processer
Dec 13th

Block Diagram

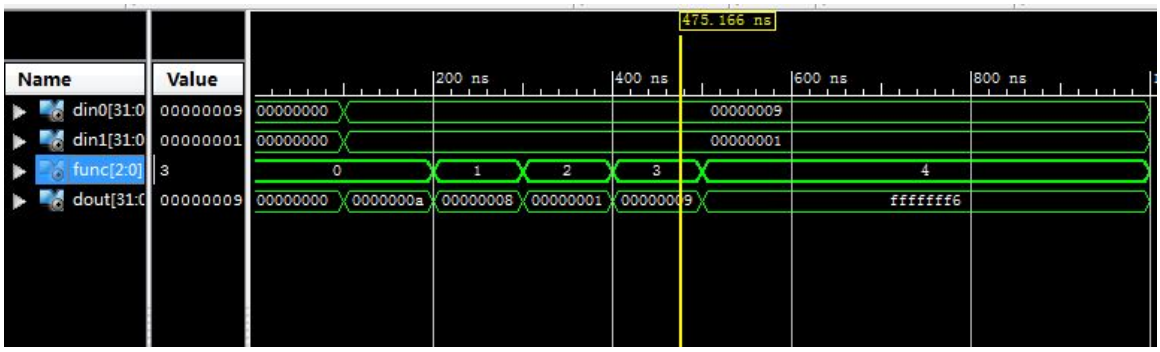
This is an overall layout of our processor design which contains 5 core modules including a program counter, a decoder, a register file, an ALU, an instruction memory and a data memory. The program counter controls the sequence of instruction and decides next instruction which will be executed next. The instruction memory stores all the instructions in pre-organized order. The decoder generates control signals for other modules depending on instructions' OP codes and function codes. The register file stores values which can be later used by the data memory and the ALU. The ALU performs all the arithmetic function including "add", "subtract", "and" and "or". The data memory stores all other values which can be loaded back to the register file.



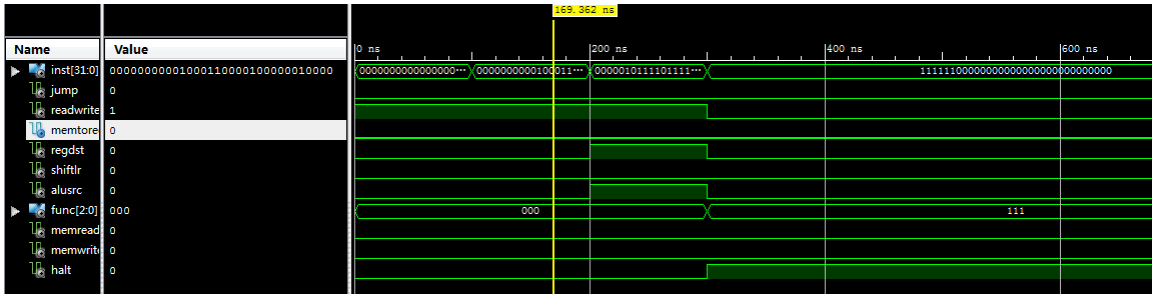
Decoder Output:
jump
readwrite
memtoreg
regdst
shift, shiftlr
alusrc
func
memread, memwrite
halt

Simulation of checkpoints

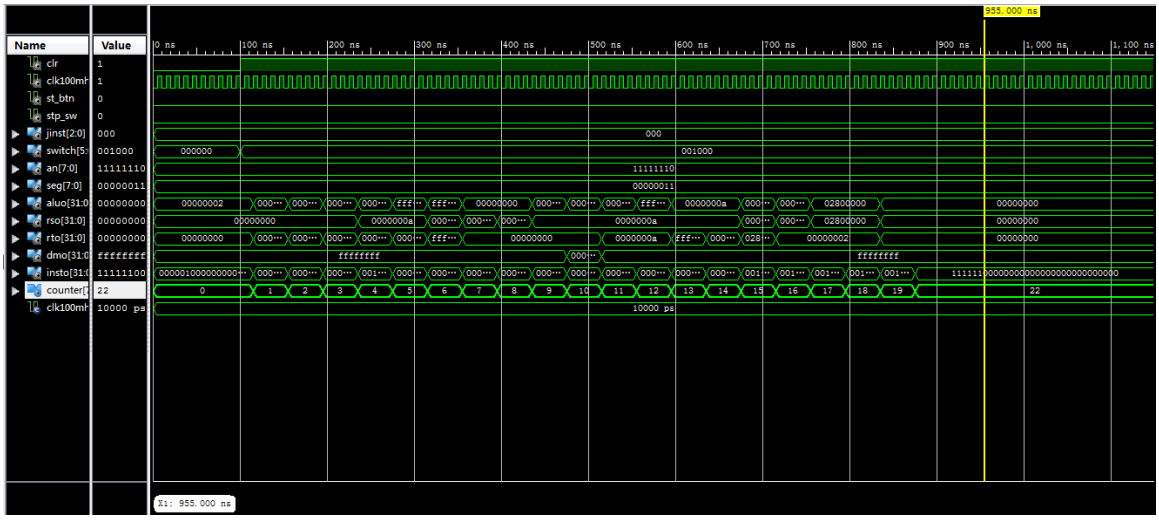
ALU Function Simulation



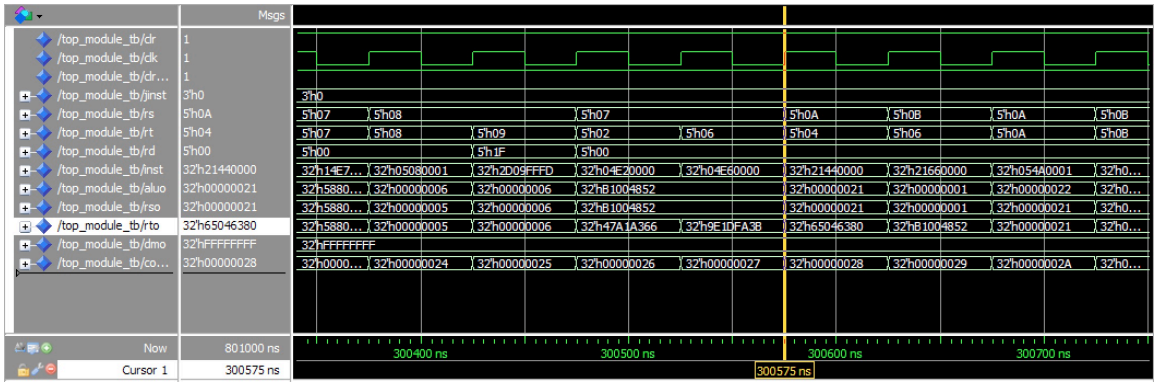
Decoder Function Simulation



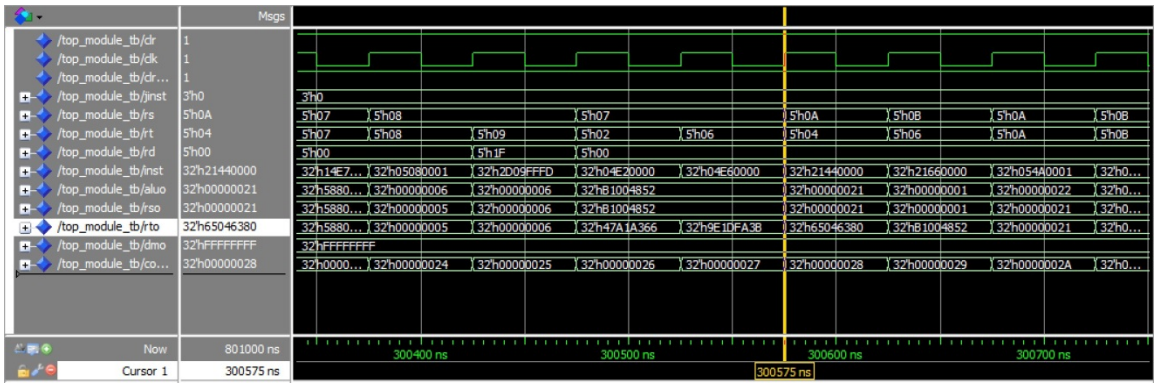
Sample Instructions Function Expansion



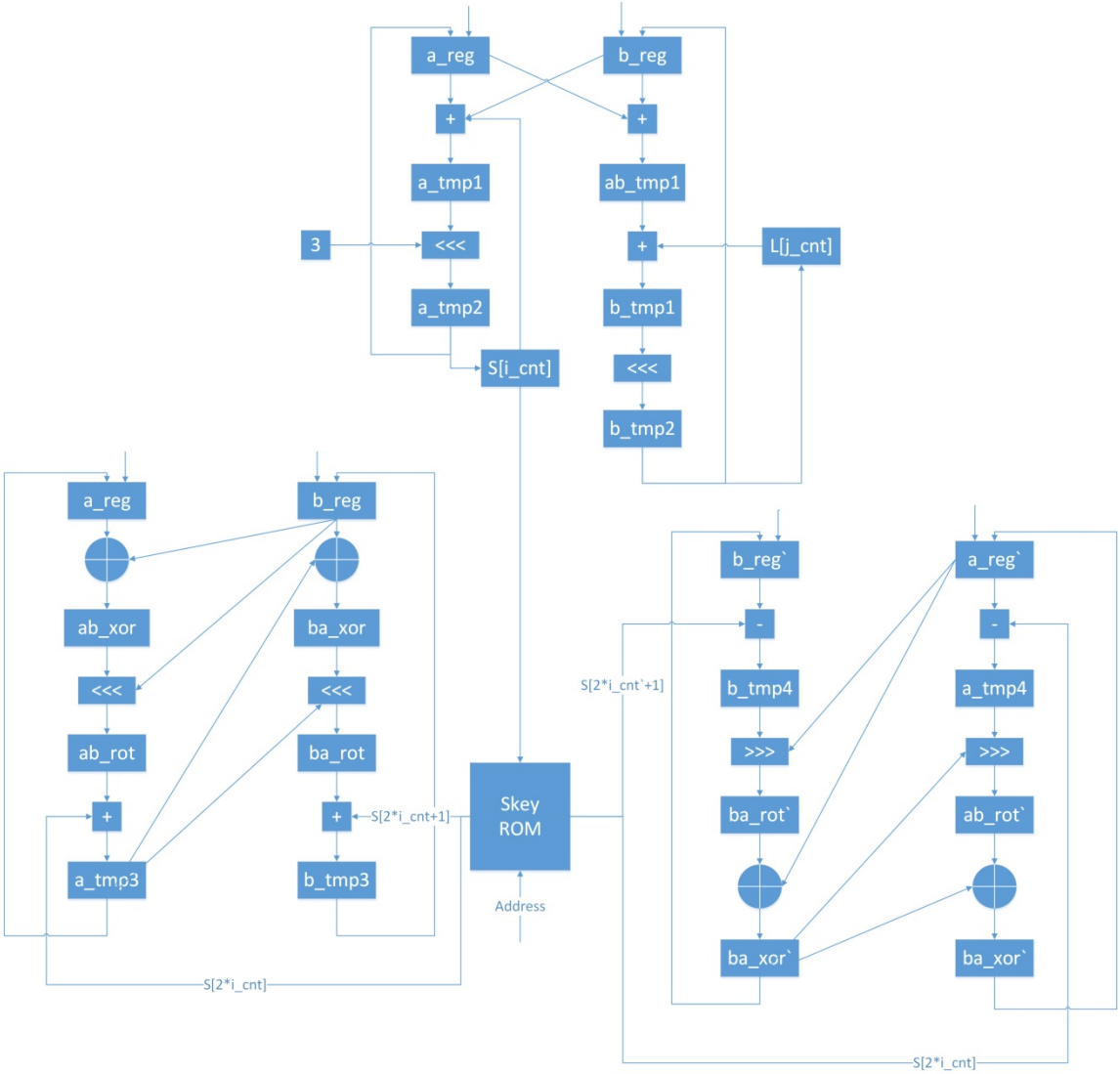
Key Expansion Function Expansion



Encryption Function Expansion



High level description of implementing and run 3 components of RC5 (encryption, decryption and key expansion) on MIPS single cycle processor (Shown in figure below)



RC5 Key Expansion in MIPS Assembly

In key-generation, firstly we set all value in register and the data memory to 0.Then we use the 0-3 in the data memory as the ukey, 8-33 in the data memory to store the value of skey0-25. There are several key points in the design. Firstly, since we don't have the instruction to shift a register by the value in another register, so we need to set a register count from 0 to the value of shift register. As the register shift by one we add to the counter by one until it equal to the value we need. Here we extract the last 5 bits of the shift value register as the shift value using AND with 1F since they are all 32-bit values, shift by that value takes long time. Another point in this design is we use several loop to make sure that the code work perfectly. In this key_exp algorithm, we have totally 78 rounds to finish, during them, every four time we must set L_array count to 0, every 26 times we must set S_array to 8, that we figure out a bne instruction to skip the line of sub if the count is not equal to that value.

RC5 Encryption in MIPS Assembly

In encryption, we define register $1 and $2 as locations of A and B. First, load the input in address of 4 and 5 into $1(A) and $2(B). There are three loops in this process. The first one is a 12-cycle encryption with counter i, which is the value in $13. We store the sum of A and S[0] in $1, and the sum of B and S[1] in $2. In a single loop 1, we load skeys (s[2*i] and s[2*i+1]) into $3 and $4. First we obtain the value of the XOR of $1 and $2, and rotate left it by the value of $2. Then we add the result with S[2*i] to get a new A, of which the value is stored in $1. Next, we obtain the value of the XOR of $2 and $1, and rotate left it by the value of $1. Then we add the result with S[2*i+1] to get a new B. When the value in %13 equals to 12, the values we get in $1 and $2 is the value of A and B. Finally, these two values are stored in the data memory.

RC5 Decryption in MIPS Assembly

In decryption, we define register $3 and $4 as locations of B and A. First, load the input in address of 6 and 7 into $3(B) and $4(A). There are three loops in this process. The first one is a 12-cycle decryption with counter i, which is the value in $12 . In a single loop 1, load skeys, s[2*i+1] and s[2*i], into $1 and $2. Then store the new value, equal to B – s[2*i+1], to B register $3. To right rotate of B by A' (in $17) value, we use a small loop to shift B by 1 bit each time until the number of shift times equal to A' (here we extract the last 5 bits of A as the shift value using AND with 1F since they are all 32-bit values). After the rotation, A XOR B is supposed to be the next. It is replaced by 5 NOR operations here. After these steps, we get a new B. A value is obtained similarly. In the big loop 1, i deceases by 1 each loop until it becomes 1 to guarantee total number of loop 1 is 12. Then, we subtract A and B by s[0] and s[1], respectively. Finally, store them into the data memory with address 36 and 37.

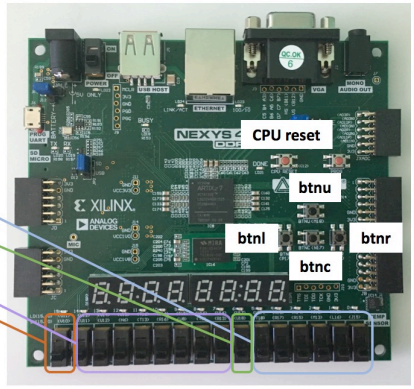
Description of processor interfaces

The default inputs for both encryption and decryption and user keys for key expansion are all 0. Those inputs are all stored in the data memory. For instance, user keys are in address 0-3 of the data memory, inputs of encryption are in address 4-5 of the data memory and inputs of decryption are in address 6-7 of the data memory.

In order to alter inputs, we add 2 instructions at the beginning of the instruction memory. This instruction is SW instruction which will store the value from the register file to the data memory. Then we add an additional multiplexer to choose data_in of the data memory from switch inputs instead of the register file. For convenience, we only implement the method to change bit 0-7 and 32-39 of encryption's inputs.

Methodology on FPGA

- Switches:
  - sw(0-5)----------to choose outputs from data memory
  - sw(6)----------to enable single stepping
  - sw(7-14)---------to change bit 0-7 and 32-39 of encryption inputs
  - sw(15)----------to choose data in of data memory between register file and switch inputs
- Buttons:
  - CPU reset-------clear
  - btnu(up)--------move instruction forward by single step
  - btnl(left)--------jump to decryption
  - btnc(central)---jump to encryption
  - btnr(right)------jump to key expansion
- LEDs:
  - led(0-15)-------show program counter
- 7-seg LEDs--------display 8 hex numbers for 32-bit binaries chosen from data memory by sw(0-5)



Instruction for inputting

- First start the FPGA in single stepping mode (switch(6)=1) and choose switches as data inputs for the data memory(switch(15)=1).
- Second change switch (7-14) to the bit 0-7 of your input value.
- Press the single stepping button (top button) to move forward by one step. (The first instruction will store the inputs value to first 8 bits at address 4 in the data memory).
- Then change switch (7-14) again to the bit 32-39 of your input value.

- Press the single stepping button to move forward by one step. (The second instruction will store the inputs value to first 8 bits at address 5 in the data memory).

Instruction for outputting

- Switches (0-5) are used to display the data memory values at corresponding location.
- LED can display values from 64 different address at the data memory. (Outputs of key expansion are stored in 8-33, outputs of encryption are stored in 34-35 and outputs of decryption are stored in 36-37)

Instruction for controlling key_exp, enc and dec

- Left, middle and right button are used for jumping to decryption, encryption and key expansion, respectively.

Performance and area analysis

We perform timing simulations for all individual modules and calculate the delay of each module. We list them below to analyze the overall critical delay which determines our max speed.

Below are delay for individual modules.

- The critical delay of ALU: 11.012 ns
- The critical delay of the decoder: 5.915 ns

We perform timing simulations for the completed processor and calculate the delay of PC, Instruction Memory and ALU. We list them below to analyze the overall critical delay which determines our max speed.

Below are delay for PC, IM and ALU on completed processor.

- The critical delay of PC: 14.565 ns
- The critical delay of IM: 16.089ns
- The critical delay of ALU: 18.159ns

The critical delay for whole processor is 18.159ns + delay of store values to the register file or the data memory which is 3.2 ns. So the total critical delay is 21.359ns.

The latency of key expansion is 7799 cycles, the latency of encryption is 1549 cycles and the latency of decryption is 1282 cycles.

The fastest speed we can run this program is (7799+1549+1282)*21.359 = 227046.

We run our FPGA using a clock divider to slow the clk down to 25ns / cycle which is slower than total critical path delay 21.359ns, so our FPGA is running properly.

Details about how you verified your overall design.

We store our key expansion, encryption and decryption instructions in instruction memory. Then we run key expansion first to generate skey for RC5. After we check the result of skey, we press jump button to jump to encryption. The encryption's output is matched with our expected value. Finally we jump to decryption, and exam the output of decryption. Because all the values of skey and outputs are correct, we concludes that the overall design is correct.

References

Karri, Ramesh. “Advanced Hardware Design.” NYU Classes. Depts. Of Electrical and Computer Engineering, New York University, 2016. Web.

Garg, Siddharth. “Computer Architechture I.” NYU Classes. Depts. of Electrical and Computer Engineering, New York University, 2016. Web.