DESIGN OF A MULTI-CYCLE AND A PIPELINED RISC-V PROCESSORS

CONTENTS

RETROSPECTIVE

- Previously I've designed a single-cycle RV32I processor with a reduced ISA of the original ISA.
- The processor was designed for educational purpose therefore there was no need to increase the ISA or include extensions like multiplication and division.
- This time I've designed two other types of the processor both support the same reduced ISA.
- (R-type, I-type, lw, sw, lui, jal, jalr, beq, blt).
- First, a Multi-Cycle microprocessor: this one is pretty simple in design It's like one big FSM machine that I will provide the state transitions diagram for later.
- Since the single-cycle design gets the entire instruction done in one cycle no matter what it is, It had to use more hardware

MULTI-CYCLE VS SINGLE-CYCLE

- For example: The current pc must be added to the value 4 each cycle to update the PC.
- As well as adding the immediate with the current PC in case the instruction was branch or jump
- So two adders must exist besides the original ALU of the processor in order to work properly
- Adders specially could be expensive too.
- A Multi-Cycle design eliminates this issue by reusing the ALU of the processor for different purposes in the same instruction.
- But by nature the multi-cycle has more cycles and more stages than a single cycle.
- So it's expected to be slower to handle more propagation and setup and hold delays.
- But of course at least in this case it simplifies the design and makes it cheaper.

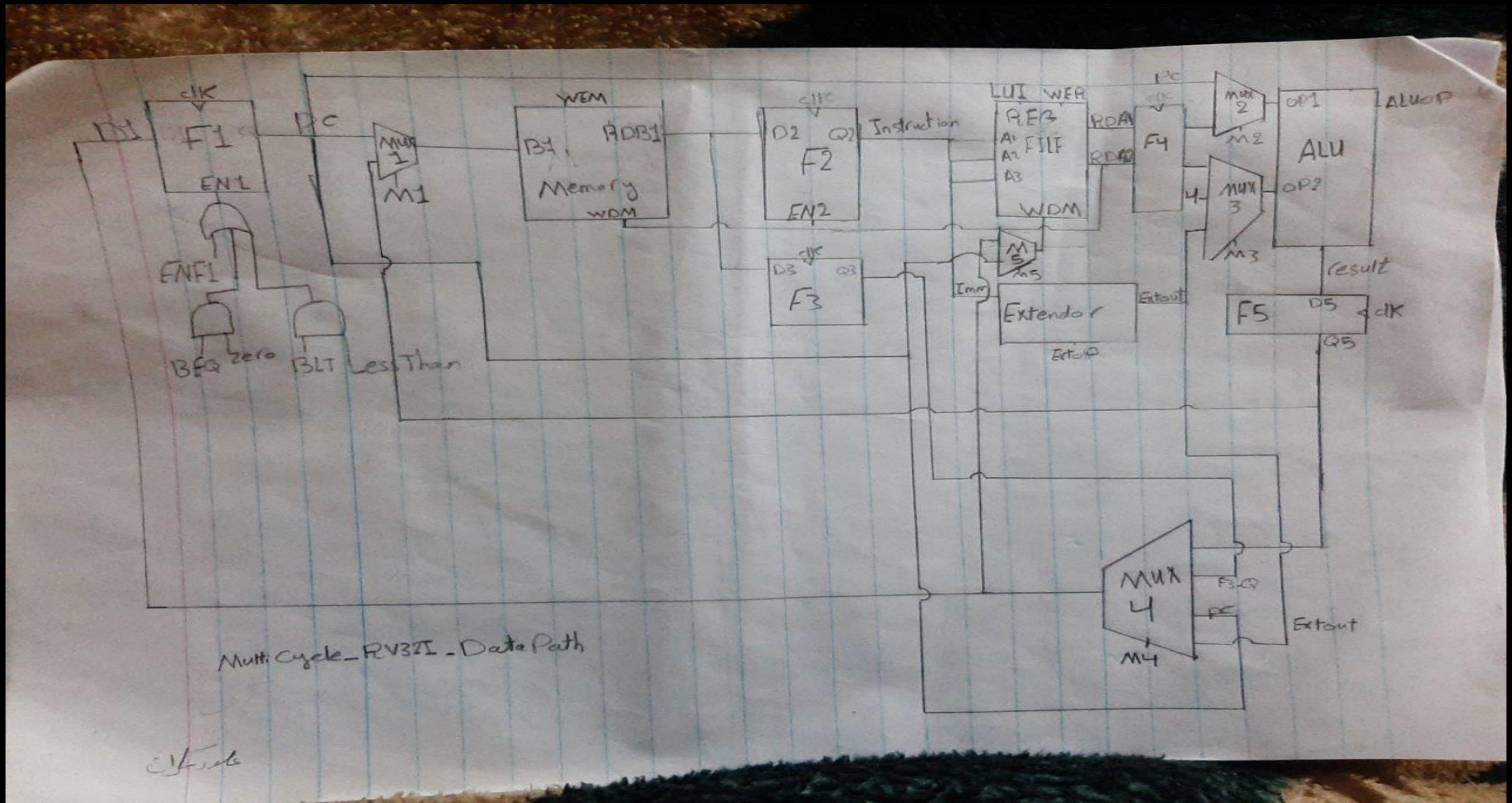
MULTI-CYCLE VS SINGLE-CYCLE

It's not always expected from Multi-Cycles to be better than single-cycle essentially It comes down to the application and the design.

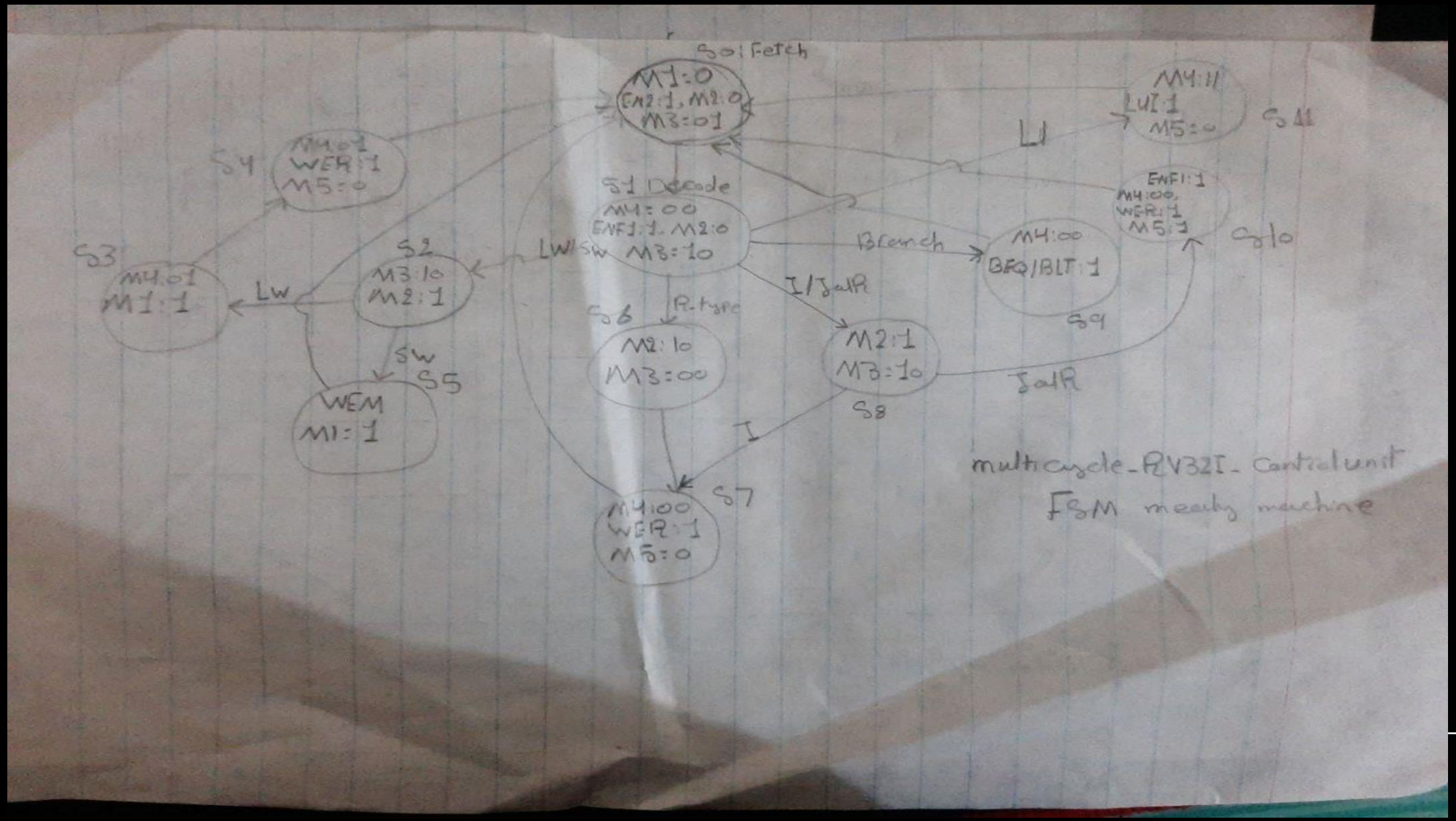
One application might require less budget than other and one design might have not so many reusable units.

Next slides include the data path of the multi-cycle design and also the control unit which is the one big FSM mealy machine that controls the flow of the CPU.

MULTI-CYCLE RV32I DATA PATH



MULTI-CYCLE RV32I CONTROL UNIT



MULTI-CYCLE RV32I DATA PATH

- we can notice the only adder in the data path is the ALU and whenever needed to add two signal it will use this unit.
- We can also see the addition of four more Flip-Flops that stage the design.
- These flops introduce propagation, setup and hold delays but they also stage the design into 4 stages.
- Each instruction fetched and decoded could utilize all stages or skip a stage if not needed.
- For example R-type instruction don't need to access memory therefore don't need to use that stage at all.

MULTI-CYCLE RV32I CONTROL UNIT

- The first two stages are constant for every instruction which are S0 (fetch) and S1 (decode)

S0 (fetch): PC goes from the Mux1 (by default select is 0 unless changed) to the instruction memory to fetch the instruction, but we can also use this moment to use another stage!

Since we're not using the ALU at this cycle we can use it concurrently to add the current PC with 4 to update the PC which is exactly what happens during this cycle.

S1 (decode): here the instruction is being decoded to determine which flow will be taken as well as accessing registers and extending immediates all happens concurrently.

Also here we're not using the ALU stage so we'll use it to add the immediate to PC in case the instruction is a branch or a jump then the jump address will be cooked ready to go later.

MULTI-CYCLE RV32I CONTROL UNIT

- The rest is easy. Each instruction takes a different flow depending on what it does.
- An instruction might take from 3 cycles to 5 cycles depending.
- Some instructions utilize the same state and some other states are exclusive to one instruction
- This state transition diagram could be changed from mealy to moore by increasing the states by one or two but there was really no point in doing so.
- In order to do later comparisons we'll compute the critical path now approximately.
- No timing analysis were done in this design but we'll just take an assumption.
- A table that has delays for different components and gates is at the end of these slides.
- Since most stages either contain unit like REGFILE or MEMORY and a mux we can already assume the critical path will be in either stage 3 or 2.

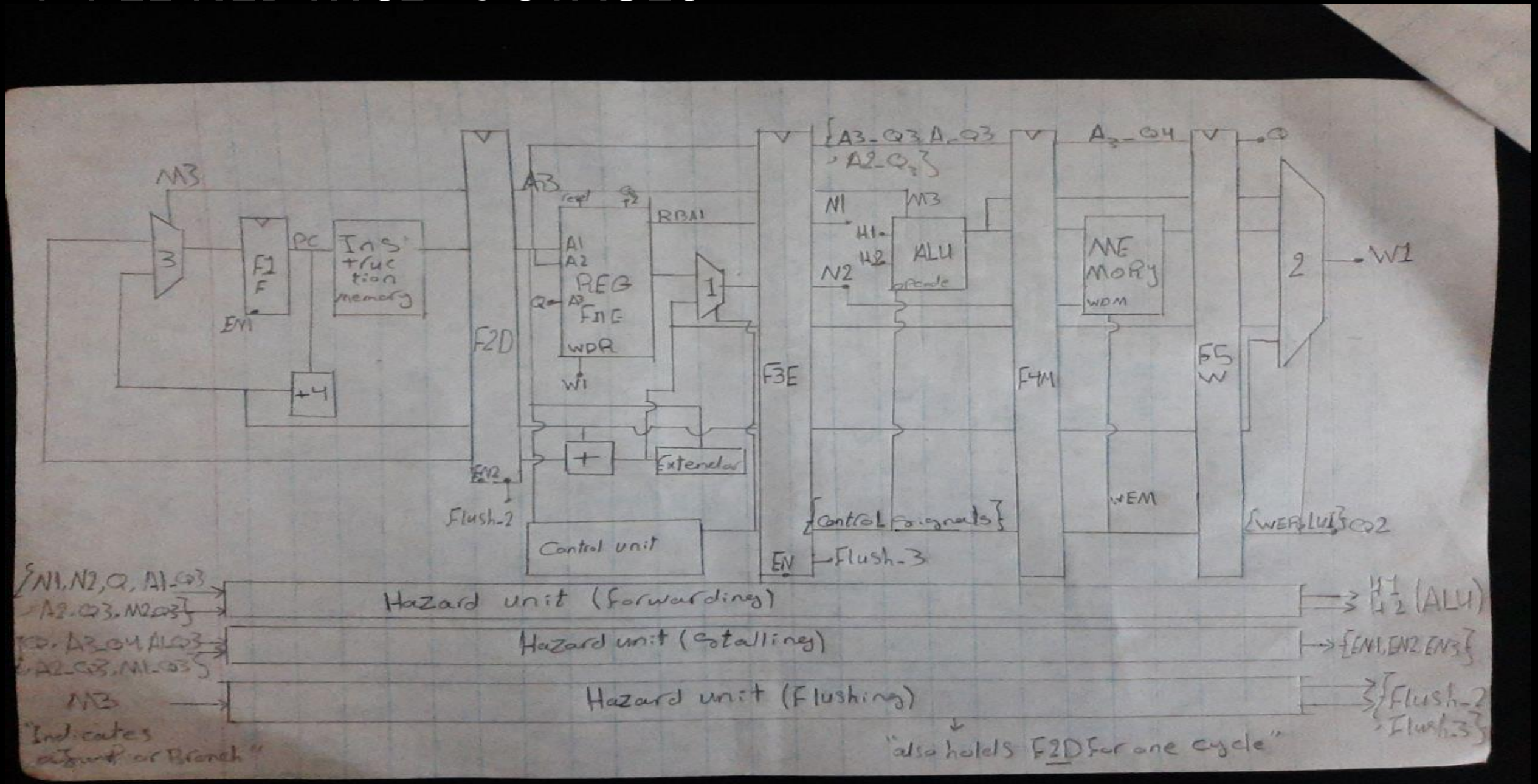
MULTI-CYCLE RV32I CRITICAL PATH

- Critical path:
- Stage 1: Flip-Flop -> Instruction memory read -> Flip-Flop:
- Critical path delay= 40+50+200=**290ps**
- That concludes our multi-cycle design the files for it will be on github

PIPELINED RV32I 5 STAGES

- Multi-Cycle won't improve the performance of a single-cycle but performance is crucial in many application like modern CPUs all of them are pipelined that is to provide higher throughput.
- Even at the cost of more registers and hardware the higher throughput of pipelining by N stages should ideally be N times the original throughput.
- Next slide have a diagram of the design.

PIPELINED RV32I 5 STAGES



PIPELINED RV32I 5 STAGES

- As opposed to multi-cycle this one doesn't save hardware in face it adds many more registers and units but way higher performance, nowadays CPUs are all pipelined and multi-core too.
- First we can see I've decided on 5 stages for the number of stages for this specific architecture It was proven that around 7 stages is the perfect number any further increase leads to more hazards and not much increase in performance.
- This was only 5 stages and the hazards were not easy to solve.
- I have installed three hazard units that are essential for proper functioning.

1- Forwarding (destination reg is source for a following instruction)

2- Stalling (must use with lw to access memory just for one cycle)

3- Flushing (must flush fetched instructions when jumping to a new address)

PIPELINED RV32I 5 STAGES

- The design wasn't time analyzed so for example:
- Ideally all 5 stages should have the same delay in between them but this system is unbalanced
- This could be improved by moving MUX1 to stage 3 and the adder in stage2 to stage 1.
- Write back writes at the falling edge this is common practice but it assumes the writing speed to the register cache is fast enough.
- Jump and branches updates the pc at the start of fourth cycle which is better than the normal write back that lags by two cycles but also increases the critical path delay.
- Critical path: from F3E to F4M (Flop to Hazard unit to ALU to 2to1 MUX to Flop)

DIFFERENT DESIGNS OF PROCESSORS

I'll assume the hazard unit delay is almost as two Muxes

Pipelined Delay=40+3(30) +120+50=300ps

This is high because of the Hazard unit (forwarding) that I accounted its delay as 2 (2to1 Muxes)

Multi-Cycle critical path delay: 290ps

Single-Cycle delay:(Flop to instruction cache to register cache to 3to1 Mux to ALU to Memory to 3to1 Mux to 2to1Mux to Flop) =40+200+100+30+120+200+30+30+50=800ps

Note that ideally the pipelined stage delay should've been 800/5=**160ps** but hazard units and flop setup and propagation delays are almost 150ps alone.

Now let's consider the SPECINT2000 benchmark which is one benchmark of many

SINGLE-CYCLE VS MULTI-CYCLE VS PIPELINED

The SPECINT2000 benchmark

consists of approximately 25% loads, 10% stores, 11% branches, 2% jumps, and 52% R- or I-type ALU instructions. Assume that 40% of the loads are immediately followed by an instruction that uses the result, requiring a stall, and that 50% of the branches are taken (mis-predicted), requiring two instructions to be flushed.

Now let's compare the execution time of 100 billion instructions.

For the single-cycle :800ps*(100 billion)=**80 seconds.**

For the multi-cycle:((0.25*5) + (0.1*4) + (0.11*5) + (0.02*5) + (0.52*5))*290*1=**142 seconds**

For pipelined: ideally it should have 1 CPI (cycle per instruction) but here there are dependencies

Load takes one cycle when there is no hazard following it but if there is it stalls for 1 cycle and then in total takes 2 cycles, same for branch takes 1 if predicted and 3 if mis-predicted.

SINGLE-CYCLE VS MULTI-CYCLE VS PIPELINED

Then average CPI for load instruction: (1*0.6)+(2*0.4)=**1.4**

CPI for branch= (0.5*3)+(0.5*1)=**2**, Jump has CPI of **3**.

remaining instructions have CPI of **1**(ideal)

Then for this program total average CPI= (0.25)(1.4) + (0.1)(1) + (0.11)(2) + (0.02) (3) + (0.52)(1) = **1.25**

Delay for pipelined 5 staged = 1.25*300pico*100billion=**37.5 seconds**

If branches and jumps writes back at cycle 5 instead of 3 then the critical path delay will be **240ps**

But also the CPI in case of a mis-prediction will be 5 then

CPI of branch/jump=(5*0.5)+(1*0.5)=**3**

In this case the execution time for the program will be 1.36.240pico.100billion=**32.64 seconds**

CONCLUSION

Then you might think it's better to make branches and jumps write back at fifth cycle then since the execution time was less.

Well, not really this benchmark has 11% branches the more this percentage increases the more mis-predictions will occur and more time will be needed for this design.

If writing PC was made to be at fifth cycle then a **branch prediction** unit will be necessary.

This conclude my design, to make it more complete one could work on:

1-**Branch prediction unit**

2-**Memory hierarchy system** (cache).

3-**Expanding the ISA** to include all other instructions like lb, sb, bgt... (shouldn't be hard).

4-**Time analysis and prototyping on a FPGA.**

5-**Adding extensions, Interrupts handling system and maybe another core.**

Next slide have a table to set the differences in designs noticed so far, all files will be on github.

CONCLUSION

	SINGLE-CYCLE	MULTI-CYCLE	PIPELINED 5-STAGE
Complexity	**Simple**	**Simple**	**Complex (hazards exist)**
Cost	**cheap**	**cheaper**	**expensive**
Performance	**Not bad (100%)**	**Worse (60%)**	**best**
CPI (cycles per instruction)	**1**	**3-5 (depending on instruction)**	**1.2 to 1.4** (depending on application)
Execution time for SPECINT2000 benchmark	**80 seconds.**	**142 seconds**	**37.5 seconds** or **32.64 seconds** in case design was slightly altered for this application
Average CPI for this benchmark	**1**	**4.9**	**1.25** or **1.36** in case design was slightly altered

DELAYS TABLE

Element	Parameter	Delay (ps)
Register clk-to-Q	t_{pcq}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{AND\text{-}OR}$	20
ALU	t_{ALU}	120
Decoder (control unit)	t_{dec}	25
Extend unit	t_{ext}	35
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

REFERENCE AND LINKS

- **Reference:**

Digital Design and Computer Architecture RISC-V Edition by Sarah Harris and David Harris.

so far this book taught me how to design the three designs plus memory hierarchy and branch predictions.

Links:

GitHub (multi-cycle): https://github.com/Alaa-Sakran/RISC-V-Multi-Cycle-Processor/tree/main

GitHub (pipelined 5-stage): https://github.com/Alaa-Sakran/Pipelined-5-stage-RISC-V-processor-RTL-design/tree/main

GitHub (Single-Cycle): https://github.com//Alaa-Sakran/SingleCycleRV32I_Processor/tree/main

LinkedIn: https://www.linkedin.com/in/alaa-sakran