Final Report Group 1

Helton Chen Claus Choy Andrew Jesson Meng Yin Tao

***Abstract*—The abstract goes here.**

I. Solution

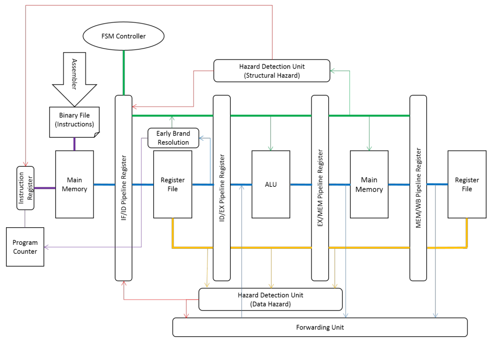

Fig. 1. Pipeline Processor Block Diagram

II. Components

The system constitutes of two components: an assembler and a processor. The assembler translates the MIPS assembly code into binary instructions. The instructions are store in the main memory along with the data. The processor accesses the memory to fetch instruction, load data and store data. The processor is pipelined into five stages and uses hazard detection, forwarding unit and early branch detection to reduce stalling time.

A. Assembler

The assembler functions by first parsing the assembly code into a structured instruction table and then converting each row of the table into binary machine language. We have implemented the assembler in MATLAB.

1) Parsing Assembly Language: To parse the assembly language the input file is first opened and read line by line. For each line comments are removed and labels are checked for. If the line is still not empty the mnemonic of the instruction and arguments are parsed and stored. After this first iteration the assembler then searches for labels in the argument fields and calculates their offset to the label position. The resulting instruction table for fib.asm is shown in Table I. Aside from the arguments and names of each instruction, an array is stored identifying the basic block that the argument belongs to. Inner blocks are stored after Arg 3 and more general blocks are stored proceeding from left to right.

TABLE I
Instruction Table Result of Parsing fib.asm

Mnemonic	Type	Arg 1	Arg 2	Arg 3	Block 1	...	Block N
addi	I	$10	$0	4			
addi	I	$1	$0	1			
addi	I	$2	$0	1			
addi	I	$11	$0	2000			
addi	I	$15	$0	4			
addi	I	$3	$2	0	loop		
add	R	$2	$2	$1	loop		
addi	I	$1	$3	0	loop		
mult	R	$10	$15		loop		
mflo	R	$12			loop		
add	R	$13	$11	$12	loop		
sw	I	$2	$13	0	loop		
addi	I	$10	$10	-1	loop		
bne	I	$10	$0	-9	loop		
beq	I	$11	$11	-1			

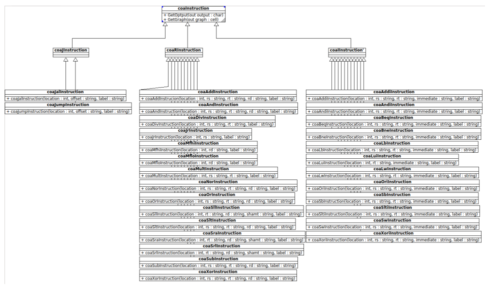

Fig. 2. Class Hierarchy of Assembler Instruction Objects

2) Converting Instruction Table to Binary Machine Language: For this procedure we take an object oriented approach. This design decision was made to support easy extension of instructions not included in the specification. The class hierarchy of the instruction API is shown in Figure 2. The public class constructors take the form of coa¡mnemonic¿Instruction(args) and return an Instruction object, I. The args for each type of instruction are taken from the above described instruction table. The public I.GetOutput() method returns the line of binary machine language corresponding to the type of instruction and arguments passed. The public I.GetGraph() method returns a structured array of the position of the instruction within the program, the register it writes to, the registers it reads from, and minimum and maximum indices of the basic block it belongs to. This method has been designed to facilitate instruction rescheduling. Table II represents the graph that would correspond to fib.asm after the initializing a static rescheduling routine with the graphs from each instruction. The routine would then maximize the distance between the true data dependencies with constraints determined by the minimum and maximum positions, along with the anti and output dependencies.

TABLE II
FIRST ITERATION DEPENDENCIES GRAPH FOR FIB.ASM

Pos.	Write	Read 1	Read 2	Min Pos.	Max Pos.	True	Anti
1	10	0		1	5		
2	1	0		1	5		
3	2	0		1	5		
4	11	0		1	5		
5	15	0		1	5		
6	3	2		6	13	3	
7	2	2	1	6	13	3	6
8	1	3		6	13	6	
9		10	15	6	12	1,5	
10	12			7	13		9
11	13	11	12	6	13	4,10	
12		13	2	6	13	11,7	
13	10	10		6	13	1	9
14		10	0	14	14	13	
15		11	11	15	15		

B. Instruction Fetch (IF) Stage

The first stage of the pipeline consists of fetching the instruction from memory and update the program counter (PC). MORE TO ADD...

C. Instruction Decode (ID) Stage

Once the instruction is fetched from memory, it is decoded in the ID stage, where the OPCODE is sent to the Finite-State Machine (FSM) controller. Depending on the current OPCODE and the historical state, the FSM controller will issue a series of control signals. Depending on the type of the MIPS instruction (R-type, I-type or J-type), the register file puts the corresponding data onto the register data line. To reduce stalling after a branch instruction, we implemented an early branch resolution unit.

– FSM Controller should probably belong here

D. EX Stage

E. MEM Stage

F. WB Stage

G. Pipeline

– Pipeline Registers
– Hazard Detection Unit
– Forwarding Unit
– Early Branch Resolution

H. Optimization

– If we are optimizing the processor, we need to discuss the design choices

III. SYSTEM EVALUATION

– How did we test it?
– What was the clock frequency?
– What was the performance?
– Dump memory and compare result against expectation?

IV. RECOMMENDATION

– Helton got this part — –intermediate steps, smaller deliverables During the first deliverable, we found it challenging when attempting to verify the results from the assembler. This is because we were not provided the expected value of the intermediate results e.g. the binary code after the assembly code is translated to bit code. The same could be said about the deliverable in general.

V. CONCLUSION

The conclusion goes here.

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to $\LaTeX$*, 3rd ed. Harlow, England: Addison-Wesley, 1999.