CSE 331 COMPUTER ORGANIZATİON

FINAL PROJECT

--HELLO MIPS—

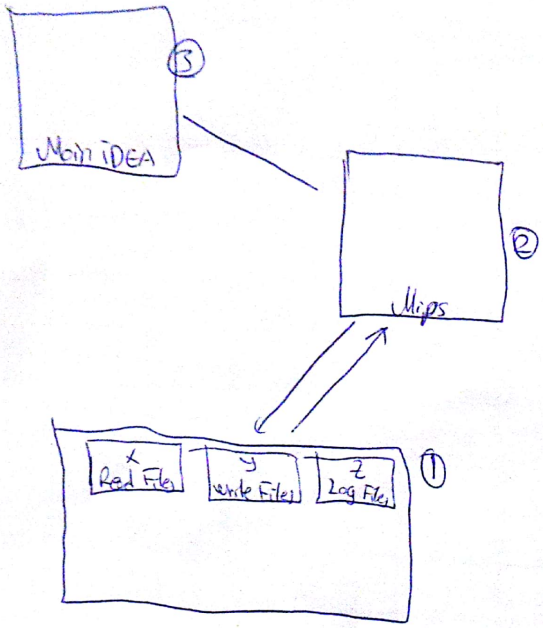
Burak Ozdemir

141044027

PROJECT REPORT

1. INTRODUCTION

1.1 Big Picture



Big Picture

① Files => Read Files : instruction.mem
: data.mem
: registers.mem
: instructionMem.txt
=> Write Files : data.mem
=> Log Files : res_registers.mem
: res-data.mem

② Mips => Basyolari ve modulleri kullanarak istenilen instructionlari isleme aliyor.

③ Main Idea : Single Cycle DataPath instructionlarini calistirabilmek

Life Cycle Of 1 Instruction

Instruction Fetch

- Instruction calistirilmak üzere instructions.mem dosyasını kullanarak alınır ve program counter 1 arttırılır. Ayrıca sinyaller üretilir.

Instruction Decode and Register Fetch

- Mux ile hedef register seçilir. Mips_registers modülü ile instruction decode edilir. I-type instructionlar için Signextend modülü ile işleme sokulur.

Execution

- ALU control unit ile sinyal üretilir. ALU Rs ve Rt or Rd registerı (tipine göre) içeriğini işleme sokar. Instruction branch ise adresi hesaplanır

Memory

- Memory unit ile gerekli işlemler alınır. Jumpadress hesaplanır ve son PC güncellenir.

Memory Read

- Load instructionı ise memory'den veri çekilip registera yazılır.

2. METHOD

2.1

- mips_instr_mem (instruction(out), PC(in)) :- Program counter'ın değeri ile instruction.mem ~~deneme~~ dosyasından ilgili instruction output olarak çıkarılır. Ayrıca tüm instructionlar registera atılır.
- Control (instruction(in), RegDst - Branch - MemRead - MemtoReg, AluOp, MemWrite AluSrc, RegWrite, Jump(out)):
  - Instruction'ın op-code kısmı ile gerekli control sinyalleri üretilir
  - PC = PC + 1
- MUX2to1_5bit (instruction(in), write_reg(in) outmux(out), RegDst) :- RegDst sinyali ile Rt or Rd register seçimi yapılır
- mips_registers (read_data_1(out), read_data_2(out), write_data(in), read_reg_1(in) read_reg_2(in), RegWrite(in), CLK(in)) :- Clock ve RegWrite sinyalinin durumuna göre register içerikleri okunur veya register'a yazılır.
- SignExtend (instruction(in), extendout(out)) :- Instruction'ın son 15 basamağı 32'ye tamamlanır. (I type)
- Mux2to1_32bit (read_data2(in), signExtendout(in), mux32out(out), AluSrc): ALU modülü için AluSrc sinyal ile read_data_2 ve signExtendout arasında seçim yapılır.
- ALU_Control (AluOp(in), instruction(in), AluSignals(out)): AluOp sinyali ile ALU için istenilen ifade için sinyal üretilir. Instruction'ın son 6 basamağına bakılır.

• ALU(ALUSignals(in), read_data_1(in), mux32out(in), AluOut(out), Zero(out)):- Gelen sinyaller ile read_data_1 ve mux32out işlene sokar. Stl instruction'ı geldiğinde Zero biti 1 olur.

— branchSelect = Branch & Zero;

• Branch Addres Calculator(signExtendout(in), PC(in), branchadress(out)): signExtendout ile PC birleştirilip branch adress hesaplanır.

• Mux2to1_32bit(PC(in), branchaddres(in), newPC(out), branchSelect(in)):- Branch ve Zero sinyallerinin durumuna göre yeni PC belirlenir.

• mips_data_mem(MemoryReadOut(out), AluOut(in), read_data_2(in), MemRead(in), MemWrite(in): Sinyallerin durumuna göre data.mem dosyasından bilgiler registerlara alınıp ya yazılır yada registerlar yazmak için okunur.

• Mux2to1_32bit(AluOut(in), MemoryReadOut(in), write_data(out), MemtoReg(in)):
- MemtoReg sinyaline göre ya memory çıkışı yada ALU çıkışı seçilir.

• JumpAddresCalc(instruction(in), ProgramCounter(in), JumpAdres(out)):- J-type instruction için jumpadress hesaplanır.

• Mux2to1_32bit(newPC(in), JumpAdrs(in), a(out), jump(in)):- Jump sinyaline göre PC yeni değeri hesaplanır.

• myassign(PC, a, clock): mips_core'da atama almadığı için PC yeni değerini başka modül üzerinde atama yaptık.

2.2 Signals

Instruction	Type	Opcode (func	RegDst	Branch	MemRead	MemWrite	AluSrc	RegWrite	Jump
add	R	100000 (f)	1	0	0	0	0	1	0
addi	I	001000	0	0	0	0	1	1	0
addiu	I	001001	0	0	0	0	1	1	0
addu	R	010101 (f)	1	0	0	0	0	1	0
and	R	011000 (f)	1	0	0	0	0	1	0
andi	I	001100	0	0	0	0	1	1	0
beq	I	000100	0	1	0	0	0	0	0
bne	I	000101	0	1	0	0	0	0	0
j	J	000010	0	0	0	0	0	0	1
jal	J	000011							
jr	R	001000 (f)							
lbu	I	100100			1				
lhu	I	100101			1				
lui	I	001111			1				
lw	I	100011			1				
nor	R	100111 (f)	1	0	0	0	0	1	0
or	R	100101 (f)	1	0	0	0	0	1	
ori	I	001101							
slt	R	100110 (f)	1	0	0	0	0	1	
slti	I	001110							
sltiu	I	001011							
sltu	R	101011 (f)	1	0	0	0	0	1	
sll	R	000000 (s)	1	0	0	0	0	1	
srl	R	000010 (s)	1	0	0	0	0	1	
sb	I	101000				1			
sh	I	101001				1			
sw	I	101011				1			
sub	R	100010 (f)	1	0	0	0	0	1	0
subu	R	100011 (f)	1	0	0	0	0	1	

ALUOP	Execute
0000	R-type
1000	add
1001	sub
1100	and
0100	or
0110	nor
0010	eq
0011	neq
1011	sleft
1010	sright
1111	sll

ALU-Control => (AluOp==0) funct field
=> (else) AluOp

ALU =>
- add, unsigned
- sub, unsigned
- and
- or
- nor
- sleft
- sright
- eq
- jr