Processor Project

CSSE232

Cullen LaKemper
Joseph Peters
Russel Staples
Will Yelton

Architecture Description

Name: CHINPO (**C**hinpo **H**ardware **I**s **N**ot **P**erfectly **O**ptimized)

About:
CHINPO is a 16-bit word, instruction, and register load store based architecture, which utilizes two preset operational registers connected to the ALU which are used for all ALU operations and temporary registers to store results. The architecture focuses on completing instructions quickly and preparing for the next operation concurrently. ALU operations such as addition and subtraction are always operated on the A and B registers and the result is placed into the destination register set through the instruction. Every non-Immediate command can concurrently move a value from a register into register A, register B, or both. Immediate commands accept an 8-bit immediate.

Registers

Number	Symbol	Description
0	$0	Zero register: Always equal to 0 cannot be changed
1	$sp	Stack Pointer: Points to the current top of the stack
2	$ra	Return Address: Points to address the current function must jump to when concluded
3	$sr	System reserved: Used for interrupts and cause etc...
4	$at	Assembler Temporary for pseudo instructions
5	$a0	Argument 0: Place and receive function arguments here
6	$a1	Argument 1: Place and receive function arguments here
7	$v0	Function Return: Place function returns here
8	$A	A: Operations register 0
9	$B	B: Operations register 1
10	$t0	Temporary register 0
11	$t1	Temporary register 1
12	$t2	Temporary register 2
13	$t3	Temporary register 3
14	$t4	Temporary register 4
15	$t5	Temporary register 5

Instruction Formats

DR Type Instructions:

4 bits	4 bits	4 bits	1 b	1 b	1 b	1 b
op	rd	rm	ma	mb	CLRa	CLRb

op	:	Operation Code	number (Defined in the table below)
rd	:	register destination	number (Addressed directly defined above)
rm	:	register to move	number (Addressed directly defined above)
ma	:	move to a	boolean (1-move, 0- do not move)
mb	:	move to b	boolean (1-move, 0- do not move)
CLRa	:	clear a	boolean (1- clear, 0- do not clear)
CLRb	:	clear b	boolean (1- clear, 0- do not clear)

I Type Instructions:

4 bits	4 bits	8 bits
op	rd	immediate

op	:	Operation code	number (Defined in the table below)
rd	:	register destination	number (Addressed directly defined above)

Instructions

DR (Double Register) Type Instructions

The register designated by rm is moved to either the A or B register as designated by ma/mb concurrently with what is described in the instruction description. Also the CLRa/CLRb bits can clear the values in A or B after an instruction is completed and before the move happens.

Syntax: inst rd, rm, ma, mb, CLRa, CLRb
Example: add $t1, $t2, 0, 1, 0, 0,

Decimal	Symbol	Name	Description
0	add	Add	Adds A to B and stores in rd
1	sub	Subtract	Subtracts B from A and stores in rd
2	and	And	Bitwise and of A and B
3	or	Or	Bitwise or of A and B
4	jr	Jump Register	Jumps to address held in A (rd not used)
5	mv	Move	Ignores the rd register
6	slt	Set Less Than	If A < B set rd to 1 else set rd to 0

I (Immediate) Type Instructions

Values are stored in the register designated by rd. The immediate does a variety of things depending on the specific instruction.

Syntax: inst rd, im
Example: beq $t0, BRANCH
lw $t1, 4

7	beq	Branch On Equal	If A == B move <immediate> instructions Beq jumps to the address defined by the (first 7 bits of the program counter + 4) + (the 8 bit immediate given shifted once)
8	lw	Load Word	The value at the address in A + (<immediate> * 2) is stored in rd
9	sw	Store Word	The value in B is stored in the address in A + (<immediate> * 2)
10	j	Jump	Jumps to tag or address PC[15-9] + <immediate> + 0

Decimal	Symbol	Name	Description
11	lli	Load Lower Immediate	Loads <immediate> into least significant bits of rd (sign extended)
12	ori	Or Immediate	Bitwise or with A and <immediate>
13	sll	Shift Left Logical	Shifts value in A by signed (immediate) and stores in rd, positive numbers shift left, negative numbers shift right
14	jal	Jump and Link	Jumps to tag or address PC[15-9] + <immediate> + 0 and stores the return address (PC+4) into $ra
15	addi	Add Immediate	Adds <immediate> to A and stores in rd

Register Transfer Language for Instructions

DR-Type	**I-Type**	**lw**	**sw**	**beq**
IR = Mem[PC] PC = PC + 4				
A= REG[1000] B = REG[1001]	A = REG[1000]		A = REG[1000] B= REG[1001]	

ALUout = A op B
if(IR[1]==1) {
REG[1000] = 0
}
if(IR[0]==1) {
REG[1001] = 0
}
if(IR[3]==1) {
REG[1000] = REG[IR[7-4]]
}
if(IR[2]==1) {
REG[1001] = REG[IR[7-4]]
}
	ALUout = A op S/E(IR[7-0])	ALUout = A + SE(IR[7-0]<<1)	ALUout = A + S/E(IR[7-0]<<1)	S/E(IR[7-0]<<1)
				IF (REG[1000] == REG[1001])
				PC = ALUout

REG[IR[11-8] = ALUout	REG[IR[11-8]] = ALUout	MDR = MEM[ALUout]	MEM[ALUout] = REG[1001]	
		REG[IR[11-8]] = MDR		

mv	**j**	**jr**	**jal**
IR = Mem[PC] PC = PC + 4			

if(IR[1]==1) {
  REG[1000] = 0
}
if(IR[0]==1) {
  REG[1001] = 0
}
if(IR[3]==1) {
  REG[1000] = REG[IR[7-4]]
}
if(IR[2]==1)
  REG[1001] = REG[IR[7-4]]
}
	PC = PC + SE(IR[7-0]<<1)	PC = REG[1000]	REG[0010] = PC
			PC = REG[1000]

RTL Component List

Name	Description	Inputs	Outputs
PC	Register that stores the program counter	**Data In (16 bits)**: The data input of the register **Clock (1 bit)**: The clock of the processor **Write (1 bit)**: Controls if data is being written or not **Reset (1 bit)**: Resets data in register to 0 if clock is enabled	**Data Out (16 bits)**: The data output of the register
IR	Instruction register		
A	Registers that input into ALU. A is always the A register and B is either the B register or an immediate		
B			
MDR	Register that stores memory data		
ALUout	Register that stores output of ALU		
REG	Register file, it has the ability to move data to the A and B registers, as well as clear them. It always outputs the data in the A and B registers.	**Rd (4 bits)**: The register address of the destination register (the register to write data to) **Write Data (16 bits)**: The data to write into the rd register **Cl A (1 bit)**: If enabled, register A will be cleared **Cl B (1 bit)**: If enabled, register A will be cleared **Mv A (1 bit)**: If enabled, the value in the Mv register will be copied to A **Mv B (1 bit)**: If	**A (16 bits)**: The data stored in the A register **B (16 bits)**: The data stored in the B register

		enabled, the value in the Mv register will be copied to B **Mv Reg (4 bits)**: The register address of the register to get data to move into A or B	
MEM	Main memory	**Write Data (16 bits):** Data to be written in main memory **Address (16 bits):** Address where the data will be written **Write (1 bit):** If enabled, the write data will be written to the address **Clock (1 bit):** The clock of the processor, data will be written with the clock is enabled	**MemData (16 bits):** Data being read from main memory
SE	Sign extend, takes the data in and copies the most significant bit 8 times to create a 16 bit value	**Data in (8 bits):** Data to be sign extended	**Data out (16 bits):** Data after being sign extended

Procedure Call Conventions

Registers

- The zero register cannot change
- sp and ra should be unchanged when returning from a procedure
- All other registers are mutable in procedures

Stack

- All mutable registers should be saved on the stack
- Extra pass in arguments should be placed in the stack at the lowest value and increase in address
- Extra return values should be placed at the highest value addresses in the stack and count down

Example

//Register A starts with an arbitrary word and B is clear
//All temp registers start with arbitrary words
//proc has four inputs and three outputs

```
ori  $a0, $t0, 1, 0, 0, 0     #put A in a0 and put t0 in A
ori  $a1, $sp, 1, 0, 0, 0     #put A in a1 and put sp in A
mv   $0,  $t1, 0, 1, 0, 0     #put t1 in B
sw   $0,  0                   #save B in A shifted 0 words
mv   $0,  $t2, 0, 1, 0, 0     #put t2 in B
sw   $0,  1                   #save B in A shifted 1 word
mv   $0,  $ra, 0, 1, 0, 0     #put ra in B
sw   $0,  2                   #save B in A shifted 2 words
jal  proc                     #jump to proc
mv   $0,  $v0, 1, 0, 0, 0     #put v0 in A
ri   $t0, $sp, 1, 0, 0, 0     #put A in t0 and put sp in A
lw   $t1, -1                  #load A shifted -1 words in t1
lw   $t2, -2                  #load A shifted -2 words in t2
lw   $ra, 3                   #load A shifted 3 words in ra
```

Code Fragments with Machine Code

Loading in a 16-bit integer:

```
lli $A, 0x16                        1011 1000 0001 0110
sll $A, 0x8                        1101 1000 0000 1000
ori $A, 0x21                        1100 1000 0010 0001
```

Results in the Register:

A: 0x1621

Loading in two numbers and adding them:

```
lli $A, 0x31                        1011 1000 0011 0001
lli $B, 0x02                        1011 1001 0000 0010
add $t0, $0, 0, 0, 1, 1             0000 1010 0000 0011
```

Results in the Register:

A: 0x0000
B: 0x0000
t0: 0x0033

Looping and iteration:

```
           add $t0, $t0, 0, 0, 1, 1   0000 1010 1010 0011
           lli $B, 0x05               1011 0101 0000 0101
Loop:      addi $A, 1                 1101 1000 0000 0001
           add $t0, $0, 0, 0, 0, 0    0000 1010 0000 0000
           beq $B, Loop               0111 1001 1111 1101
           add $A, $0, 0, 0, 0, 1     0000 1000 0000 0001
```

Results in the Registers:

A: 0x000A
B: 0x0000
t0: 0x0028

Euclid's Algorithm:

```
31  relPrime:
32              # n is already in $a0 from where this was called
33              lli $a1, 2                # store m in a1                                  1011 0101 0000 0010
34  loop:       lli $B, 4                 # load 4 into B                                  1011 1001 0000 1000
35              mv $0, $sp, 1, 0, 0, 0    # move sp into A                                 0101 0000 0001 1000
36              sub $sp, $a1, 0, 1, 0, 0  # decrease sp by 4 and move $a1 into $B        0001 0001 0110 0100
37              mv $0, $sp, 1, 0, 0, 0    # move the value in $sp into $A                  0101 0000 0001 1000
38              sw $0, 0                  # stores m on the stack                          1001 0000 0000 0000
39              mv $0, $a0, 0, 1, 0, 0    # moves n to $B                                  0101 0000 0101 0100
40              sw $0, 1                  # stores n on the stack                          1001 0000 0000 0100
41              mv $0, $ra, 0, 1, 0, 0    # move $ra into B                                0101 0000 0010 0100
42              sw $0, 2                  # store $ra on the stack                         1001 0000 0000 1000
43              jal gcd                   # jump into the gcd function                     1110 0000 (address of gcd)
44              mv $0, $sp, 1, 0, 0, 1    # put sp into $A                                 0101 0000 0001 1000
45              lw $a0, 1                 # load n back into $a0                           0100 0101 0000 0100
46              lw $a1, 0                 # load m back into $a1                           0100 0110 0000 0000
47              lw $ra, 2                 # load ra back                                   0100 0010 0000 1000
48              lli $A, 3                 # put 4 into A                                   1011 1000 0000 1000
49              add $sp, $0, 0, 0, 0, 0   # add 4 back to the stack pointer               0000 0001 0000 0000
50              mv $0, $v0, 1, 0, 0, 0    # put the result of gcd into $A                  0101 0000 0111 1000
51              lli $B, 1                 # put 1 into $B                                  1011 1001 0000 0001
52              beq $0, INCREMENT         # if result == 1, loop                           0111 0000 0000 0010
53              mv $0, $a1, 1, 0, 0, 1    # move $a1 into A and clear B                    0101 0000 0110 1001
54              add $v0, $0, 0, 0, 0, 0   # put m into $v0 to return                       0000 0111 0000 0000
55              j DONE                    # if result != 1, then return m                  1010 0000 (address of DONE)
56  INCREMENT:
57              add $a1, $0, 0, 0, 0, 0   # add 1 to m in $a1                             0000 0110 0000 0000
58              j LOOP                    # jump to loop                                   1010 0000 (address of LOOP)
59
```

```
60  gcd:
61              mv $0, $a0, 1, 0, 0, 1    # move $a0 into A and clear B                    0101 0000 0101 1001
62              beq $0, RETURNB           # if a == 0, return b                            0111 0000 0000 1010
63
64  LOOP2:      mv $0, $a1, 0, 1, 1, 0    # move $a1 into B and clear A                    0101 0000 0110 1000
65              beq $0, RETURNA           # if b == 0, return a                            0111 0000 0000 1010
66              mv $0, $a0, 1, 0, 0, 0    # move $a0 back into A                           0101 0000 0101 1000
67              slt $t0, $0, 0, 0, 0, 0   # check if a < b                                 0110 1010 0000 0000
68              beq $0, ELSE              # if a !< b go to the else                       0111 0000 0000 0010
69              sub $a0, $0, 0, 0, 0, 0   # a = a - b                                      0001 0101 0000 0000
70              j LOOP2                                                                    1010 0000 (address of LOOP2)
71  ELSE:
72              mv $0, $a0, 0, 1, 0, 0   # move $a0 into B                               0101 0000 0101 0100
73              mv $0, $a1, 1, 0, 0, 0   # move $a1 into A                               0101 0000 0110 1000
74              sub $a1, $0, 0, 0, 0, 0   # b = b - a                                     0001 0110 0000 0000
75              j LOOP2                                                                    1010 0000 (address of LOOP2)
76
77  RETURNB:
78              mv $0, $a1, 1, 0, 0, 1   # move $a1 into A and clear B                    0101 0000 0110 1001
79              j DONE                                                                     1010 0000 (address of DONE)
80  RETURNA:
81              mv $0, $a0, 1, 0, 0, 1   # move $a0 into A and clear B                    0101 0000 0101 1001
82  DONE:
83              mv $0, $ra, 1, 0, 0, 0   # move $ra into A
84              jr $0, $0, 0, 0, 0, 0    # jump to the return address in A                0100 0000 0000 0000
```