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 Bytes	4 Bytes	4 Bytes	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 Bytes	4 Bytes	8 Bytes
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 rd 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
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

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

Code Fragments with Machine Code

Loading in a 16-bit integer:

```
lui $A, 0x16                        1011 1000 0001 0110
ori $A, 0x21                        1100 1000 0010 0001
```

Results in the Register:

A: 0x1621

Loading in two numbers and adding them:

```
li $A, 0x31                         1010 1000 0011 0001
li $B, 0x02                         1010 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:

```
           li $B, 0x05              1010 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:

```
relPrime:
            # n is already in $a0 from where this was called
            lli $a1, 2                # store m in a1                              1011 0101 0000 0010
loop:       lli $B, 12                # load 4 into B                              1011 1001 0000 1000
            mv $0, $sp, 1, 0, 0, 0    # move sp into A                             0101 0000 0001 1000
            sub $sp, $a1, 0, 1, 0, 0  # decrease sp by 8 and move $a1 into $B      0001 0001 0110 0100
            mv $0, $sp, 1, 0, 0, 0    # move the value in $sp into $A              0101 0000 0001 1000
            sw $0, 0                  # stores m on the stack                      1001 0000 0000 0000
            mv $0, $a0, 0, 1, 0, 0    # moves n to $B                              0101 0000 0101 0100
            sw $0, 1                  # stores n on the stack                      1001 0000 0000 0100
            mv $0, $ra, 0, 1, 0, 0    # move $ra into B                            0101 0000 0010 0100
            sw $0, 2                  # store $ra on the stack                     1001 0000 0000 1000
            jal gcd                   # jump into the gcd function                 1110 0000 (address of gcd)
            mv $0, $sp, 1, 0, 0, 0    # put sp into $A                             0101 0000 0001 1000
            lw $a0, 1                 # load n back into $a0                       0100 0101 0000 0100
            lw $a1, 0                 # load m back into $a1                       0100 0110 0000 0000
            lw $ra, 2                 # load ra back                               0100 0010 0000 1000
            lli $A, 3                 # put 3 into A                               1011 1000 0000 1000
            add $sp, $0, 0, 0, 0, 0   # add 3 back to the stack                    0000 0001 0000 0000
            mv $0, $v0, 1, 0, 0, 0    # put the result of gcd into $A              0101 0000 0111 1000
            lli $B, 1                 # put 1 into $B                              1011 1001 0000 0001
            beq $0, INCREMENT         # if result == 1, loop                       0111 0000 0000 0010
            mv $0, $a1, 1, 0, 0, 1    # move $a1 into A and clear B                0101 0000 0110 1001
            add $v0, $0, 0, 0, 0, 0   # put m into $v0 to return                   0000 0111 0000 0000
            j DONE                    # if result != 1, then return m              1010 0000 (address of DONE)
INCREMENT:
            add $a1, $0, 0, 0, 0, 0   # add 1 to m in $a1                          0000 0110 0000 0000
            j LOOP                    # jump to loop                               1010 0000 (address of LOOP)
```

```
gcd:
            mv $0, $a0, 1, 0, 0, 1    # move $a0 into A and clear B                0101 0000 0101 1001
            beq $0, RETURNB           # if a == 0, return b                        0111 0000 0000 1010

LOOP2:      mv $0, $a1, 0, 1, 1, 0    # move $a1 into B and clear A                0101 0000 0110 1000
            beq $0, RETURNA           # if b == 0, return a                        0111 0000 0000 1010
            mv $0, $a0, 1, 0, 0, 0    # move $a0 back into A                       0101 0000 0101 1000
            slt $t0, $0, 0, 0, 0, 0   # check if a < b                             0110 1010 0000 0000
            beq $0, ELSE              # if a !< b go to the else                   0111 0000 0000 0010
            sub $a0, $0, 0, 0, 0, 0   # a = a - b                                  0001 0101 0000 0000
            j LOOP2                                                                1010 0000 (address of LOOP2)
ELSE:
            mv $0, $a0, 0, 1, 0, 0   # move $a0 into B                             0101 0000 0101 0100
            mv $0, $a1, 1, 0, 0, 0   # move $a1 into A                             0101 0000 0110 1000
            sub $a1, $0, 0, 0, 0, 0  # b = b - a                                   0001 0110 0000 0000
            j LOOP2                                                                1010 0000 (address of LOOP2)

RETURNB:
            mv $0, $a1, 1, 0, 0, 1   # move $a1 into A and clear B                 0101 0000 0110 1001
            j DONE                                                                 1010 0000 (address of DONE)
RETURNA:
            mv $0, $a0, 1, 0, 0, 1   # move $a0 into A and clear B                 0101 0000 0101 1001
DONE:
            mv $0, $ra, 1, 0, 0, 0   # move $ra into A
            jr $0, $0, 0, 0, 0, 0    # jump to the return address in A             0100 0000 0000 0000
```