Jorge Agüero Zamora
Arturo Chinchilla Sánchez
Crisptofer Fernández Fernández

CE-4301 Arquitectura de Computadores I
Tecnológico de Costa Rica
Reference Data Sheet

CORE INSTRUCTION SET — **Based on the MIPS architechture**

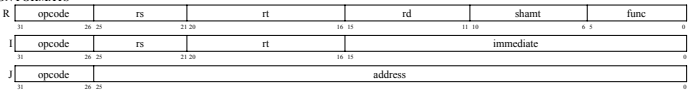
NAME	MNEMONIC	FORMAT	OPERATION	CODE/FUNCT	PURPOSE
Add	add	R	R[rd] = R[rs] + R[rt]	0 / 1	Basic math operation
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	1	Basic math operation
And	and	R	R[rd] = R[rs] & R[rt]	0 / 2	Basic logic operation
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	3	Allows change of the flow of the program
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	4	Allows change of the flow of the program
Jump	j	J	PC=JumpAddr	5	Allows change of the flow of the program
Jump And Link	jal	J	R[$ra]=PC+8;PC=JumpAddr	6	Allows change of the flow of the program
Jump Register R	jr	R	PC=R[rs]	24	
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	7	Loading data from memory
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])	0/3	Basic logic operation
Or	or	R	R[rd] = R[rs] | R[rt]	0/4	Basic logic operation
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	0/5	Inequality operation to compare values
Set Less Than Immediate	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	9	Inequality operation to compare values
Shift Left Logical	sll	R	R[rd] = R[rs] << shamt	0/6	Basic logic operation
Shift Right Logical	srl	R	R[rd] = R[rs] >> shamt	0/7	Basic logic operation
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	10	Store data to memory
Substract	sub	R	R[rd] = R[rs] - R[rt]	0/8	Basic math operation

SignExtImm = { 16{immediate[15]}, immediate }
ZeroExtImm = { 16{1'b'0}, immediate }
BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
JumpAddr = { PC+4[31:28], address, 2'b0 }
shamt = shift amount

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION	PURPOSE
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label	Allows change of the flow of the program
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label	Allows change of the flow of the program
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label	Allows change of the flow of the program
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label	Allows change of the flow of the program
Load Immediate	li	R[rd] = immediate	Load a value to a register
Move	move	R[rd] = R[rs]	Easily copy value from one register to another

BASIC INSTRUCTION FORMATS



REGISTER ASSIGNMENT

Name	Number	Purpose
$black	0	Black color Register
$cr	1	Color Register
$v0-$v1	2-3	Values for Function Results and Expression Evaluation
$hr	4	Horizontal resolution
$vr	5	Vertical resolution
$a0-$a1	6-7	Arguments
$t0-$t8	8-16	Temporaries
$s0-$s7	17-24	Saved Temporaries
$t9-$t10	25-26	Temporaries
$pp	27	Pixel Pointer
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Buffer Pointer
$ra	31	Return Address

ADDED INSTRUCTIONS

NAME	MNEMONIC	FORMAT	OPERATION	CODE	PURPOSE
Move pixel pointer	mpp	R	R[$pp] = R[$fp]+R[rt]*(R[$hr]<<2)+R[rs]<<2	12	Saves the address in the pixel pointer ($pp) of the pixel corresponding to the coordinates x,y where x=rs and y=rt
Explanation: The idea behind this instruction is to allow the programer to easily refer to pixels in an (x,y) coordinate system corresponding to the pixels on the screen and get the address of the frame buffer corresponding to that pixel, therefore this address works as a pointer to be able to paint pixels by writing to it					
Move $pp Immediate	mppi	J	R[$pp] = R[$fp] + address[25:16] * R[$hr] <<2+ address[15:0] <<2	13	Saves the address in the pixel pointer ($pp) of the pixel corresponding to the coordinates (x,y) where x = address[15:0] and y = address[25:16]
Explanation: This instruction allows the programmer to provide (x,y) coordinates for a pixel and save in the pixel pointer the address corresponding to that pixel in the frame buffer					
Paint Pixel	ppxl	I	M[R[$pp]] = R[rt]	14	Saves the color value in rd in the memory address corresponding to the current pixel pointed by $pp
Explanation: Allows the programmer to easily store the value in rs to the memory address pointed to by $pp					
Paint and Move Up	ptmu	I	M[R[$pp]] = R[$cr] ; R[$pp] = R[$pp] - R[$hr]<<2	15	Paints current pixel pointed by $pp and moves $pp one pixel up
Explanation: Allows the programmer to easily paint the pixel pointed to by $pp with value in rs, and move $pp up by one pixel up vertically					
Paint and Move Left	ptml	I	M[R[$pp]] = R[$cr] ; R[$pp] = R[$pp] - 4	16	Paints current pixel pointed by $pp and moves $pp one pixel left
Explanation: Allows the programmer to easily paint the pixel pointed to by $pp with value in rs, and move $pp up by one pixel left horizontally					
Paint and Move Down	ptmd	I	M[R[$pp]] = R[$cr] ; R[$pp] = R[$pp] + R[$hr]<<2	17	Paints current pixel pointed by $pp and moves $pp one pixel down
Explanation: Allows the programmer to easily paint the pixel pointed to by $pp with value in rs, and move $pp up by one pixel down vertically					
Paint and Move Right	ptmr	I	M[R[$pp]] = R[$cr] ; R[$pp] = R[$pp] + 4	18	Paints current pixel pointed by $pp and moves $pp one pixel right
Explanation: Allows the programmer to easily paint the pixel pointed to by $pp with value in rs, and move $pp up by one pixel right horizontally					
Paint Pixel	ppxlc	I	M[R[$fp]+R[rt]*(R[$hr]<<2)+R[rs]<<2] = R[$cr]	19	Saves the color value stored in $cr register to the memory address corresponding to pixel in coordinates x,y where x=rs and y=rt
Explanation: The idea for this instruction is to allow the programmer to easily paint a pixel on the screen providing the (x,y) coordinates in the rs and rt registers, and the value of the pixel they want to write on the rd register					

Paint and move Pixel	pmpxl	R	R[$pp] = R[$fp]+R[rt]*(R[$hr]<<2)+R[rs]<<2 M[$pp] = R[$cr];	20	Saves the color value stored in $cr register to the memory address corresponding to pixel in coordinates x,y where x=rs and y=rt, and sets $pp to the calculated address

Explanation: The idea for this instruction is to allow the programmer to easily paint a pixel on the screen and move the pixel pointer

Clear screen	cs	I	[M[R[$fp]], M[R[$fp]+R[$hr]*R[$vr]<<2]] = R[$black]	21	Set the frame buffer to zero to clear the screen

Explanation: The idea of this instruction is to allow the programmer to easily clear the screen, this instruction will set the memory appointed to the frame buffer to x0000 which is black

PSEUDOINSTRUCTIONS

Small point	spnt	I	Draws a circle 3 pixels wide centered in the pixel pointed by R [$pp] with color value specified in R[rs]	22	Graphic primitive. Translates to several move and paint pixel instructions

Explanation: This instruction provides a primitive graphic element to draw. It will save the value of the register provided in rs (the color) to the address provided in $pp (the pixel pointer) corresponding to the center of the point, from there it will also save this value to the addresses corresponding to pixels sorrounding this address (See Figure 1) This instruction needs to make 9 memory accesses to paint the point and the calculations for those addresses

Medium point	mpnt	I	Draws a circle 7 pixels wide centered in the pixel pointed by R [$pp] with color value specified in R[rs]	23	Graphic primitive. Translates to several move and paint pixel instructions

Explanation: This instruction provides a primitive graphic element to draw. It will save the value of the register provided in rs (the color) to the address provided in $pp (the pixel pointer) corresponding to the center of the point, from there it will also save this value to the addresses corresponding to pixels sorrounding this address (See Figure 2). This instruction needs to paint 37 pixels which means 37 memory accesses and the address calculations

SUPPORTED DATA TYPES

TYPE	SIZE (Bytes)
int	4
pixel	4

Pixel order

R[23,16],G[15,8],B[7,0]

Data Alignment

32 bit word

All data is word aligned since data access instructions only access 1 word

Big Endian

Memory Addressing

As the architecture is based on MIPS it is a load-store RISC architechture

Register and immediate addressing for ALU instructions

Displacement addressing (register + immediate) used in load and store operations simulates register indirect and immediate addressing and allows full 32 bit addressing by register

PC Relative addressing for branch and jump instructions

Image settings

Resolution	320x240
Pixels	76800
Total Bytes (32 bit pixels)	307200

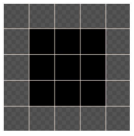


Figure 1. 3x3 small point

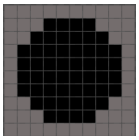


Figure 2. 7x7 medium point

The next MIPS assembly code correspond to the expantion of the small point painting code routine

```
#Small point spnt
#$rt equal to pixel color

ptmd $rt #Paint and move down
ptml $rt #Paint and move left
ptmu $rt #Paint and move up
ptmu $rt #Paint and move up
ptmr $rt #Paint and move rigth
ptmr $rt #Paint and move rigth
ptmd $rt #Paint and move down
ptmd $rt #Paint and move down
ptml $rt #Paint and move left
sll $v0, $hr, 2 #Multiply by 4
sub $pp, $pp, $v0 #Move pixel pointer to up
```

The next MIPS assembly code correspond to the expantion of the medium point painting code routine

```
#Medium pointer instruction mpnt
#$rt equal to pixel color
#Store the original value of $pp
move $v0, $pp

#Move pixel pointer to left upper small point center
addi $pp, $pp, -4 #Pixel pointer minus 4
sll $v1, $hr, 2 #Load $hr value intort and multiply by 4
sub $pp, $pp, $v1 #Move pixel pointer to up

#$v1 = R[$hr]<<2

spnt $rt  #Paint left upper small point
sub $pp, $pp, $v1 #Move pixel pointer to up
sub $pp, $pp, $v1 #Move pixel pointer to up
```

```
#Paint upper bound
ptmr $rt #Paint and move rigth
ptmr $rt #Paint and move rigth
ptmr $rt #Paint and move rigth
ptml $rt #Paint and move down
add $pp, $pp, $v1 #Move pixel pointer to down

spnt $rt #Paint rigth upper small point
addi $pp, $pp, 8 #Move 2 pixels to rigth

#Paint rigth bound
ptmd $rt #Paint and move down
ptmd $rt #Paint and move down
ptmd $rt #Paint and move down
ptml $rt #Paint and move left

addi $pp, $pp, -8 #Move pixel pointer one space to left
spnt $rt #paint rigth lower small point
add $pp, $pp, $v1 #Move pixel pointer to down
add $pp, $pp, $v1 #Move pixel pointer to down

#Paint lower bound

ptml $rt #Paint and move left
ptml $rt #Paint and move left
ptml $rt #Paint and move left
ptmu $rt #Paint and move up
sub $pp, $pp, $v1 #Move pixel pointer to up


spnt $rt #paint left lower small point

addi $pp, $pp, -8 #Move pixel pointer 2 spaces to left

#Paint left bound


ptmu $rt #Paint and move up
ptmu $rt #Paint and move up
ptmu $rt #Paint and move up
ptmr $rt #Paint and move rigth

#Reestore the original $pp value
move $pp, $v0
```