Data-sheet

Omar Amer

August 2021

0.1 General Description

RISC-Like pipelined general purpose processor with dedicated graphics and sound units with 8 32-bit registers and 128 Mb of ram with integrated VGA support and ICSP, programmed using custom, feature rich assembly.

0.1.1 Ports

Name	Width	Description
Input	32-bit	User input port
Output	32-bit	Output port
CLK	1-bit	Clock
RST	1-bit	Asynchronous reset

0.1.2 Registers

Register	Description
AX	Accumulator Register
BX	Base Register
CX	Counter Register
DX	Data Register
GX	Graphics Register
EX	Extra Register
EY	Extra Register
EZ	Extra Register

0.2 Instruction Set

Instruction	**Description**
NOP	No operation
MOV dst, src/imm	Moves data from src/imm to dst
NOT reg	Bit-wise NOT
AND dst, src/imm	Bit-wise AND, result stored in dst.
OR dst, src/imm	Bit-wise OR, result stored in dst
XOR dst, src/imm	Bit-wise XOR, result stored in dst
XNOR dst, src/imm	Bit-wise XNOR, result stored in dst
NOR dst, src/imm	Bit-wise NOR, result stored in dst
NAND dst, src/imm	Bit-wise NAND, result stored in dst
SLL reg, imm	Logical shift LEFT, shifts ***reg*** content by ***imm***.
SRL reg, imm	Logical shift RIGHT, shifts ***reg*** content by ***imm***.
INC reg	Increment reg
DEC reg	Decrement reg
OUT reg/imm	Output lower 8-bits of reg on port
IN reg	Take input from port and place data into reg
ADD dst, src/imm	ADD src/imm + dst, result stored in dst
SUB dst, src/imm	SUB dst - src/imm, result stored in dst
MUL dst, src/imm	MUL src/imm * dst, result stored in EX:EY
DIV dst, src	DIV dst/src, result stored in dst
PUSH reg/imm	Push reg/imm onto stack
POP reg	Pops stack top into reg
CMP reg1, reg2	Performs reg1 - reg2, without changing register values
JMP [addr]	Unconditional jump to [addr]
JZ [addr]	Jump if zero flag = 1 to [addr]
JNZ [addr]	Jump if zero flag = 0 to [addr]
JEQ [addr]	Jump if *equal*, i.e: zero flag = 1 to [addr]
JNEQ [addr]	Jump if *not equal*, i.e: zero flag = 0 to [addr]
JG [addr]	Jump if *greater than*, i.e: sign flag = 0 & zero flag = 0 to [addr]
JL [addr]	Jump if *less than*, i.e: sign flag = 1 zero flag = 0 to [addr]
JGE [addr]	Jump if *greater than or equal*, i.e: sign flag = 0 or zero flag = 1 to [addr]
JLE [addr]	Jump if *less than or equal*, i.e: sign flag = 1 or zero flag = 1 to [addr]
LD reg, [addr]	Load data from memory [addr] to reg
ST reg, [addr]	Store data to memory [addr] from reg
GRA	?????
SND	?????

0.2.1 Instruction Word

Type A Instructions (One / Zero Operand)

By hex instruction, we mean the full range of possible operand combinations for the given op code. for example, the op code *31 means* ***PUSH BX****, 30 means* ***PUSH AX****, and 3F means* ***PUSH EZ***

Instruction	***OP Code (binary)***	***reg***	***Hex Instruction***
NOP	*000000*		*00:07*
NOT reg	*000001*		*08:0F*
INC reg	*000010*		*10:17*
DEC reg	*000011*	*(000:111)*	*18:1F*
IN reg	*000100*		*20:27*
OUT reg	*000101*		*28:2F*
PUSH reg	*000110*		*30:37*
POP reg	*000111*		*38:3F*

Type B Instructions

Instruction	***OP Code***	***Operand 1***	***Operand 2***	***Hex Instruction***
MOV dst, src	*001000*	*(000:111)*	*(000:111)*	*200:23F*
UNUSED	*001001*			
CMP reg1, reg2	*001010*	*(000:111)*	*(000:111)*	*280:2BF*
AND dst, src	*001011*	*(000:111)*	*(000:111)*	*2CO:2FF*
OR dst, src	*001100*	*(000:111)*	*(000:111)*	*300:33F*
XOR dst, src	*001101*	*(000:111)*	*(000:111)*	*340:37F*
XNOR dst, src	*001110*	*(000:111)*	*(000:111)*	*380:3BF*
NOR dst, src	*001111*	*(000:111)*	*(000:111)*	*3CO:3FF*
NAND dst, src	*010000*	*(000:111)*	*(000:111)*	*400:43F*
ADD dst, src	*010001*	*(000:111)*	*(000:111)*	*440:47F*
SUB dst, src	*010010*	*(000:111)*	*(000:111)*	*480:4BF*
MUL dst, src	*010011*	*(000:111)*	*(000:111)*	*4CO:4FF*
DIV dst, src	*010100*	*(000:111)*	*(000:111)*	*500:53F*

Type C Instructions

Instruction	*OP Code*	*Operand 1*	*Operand 2*	*Hex Instruction*
MOV reg, imm32	*010101*	*(000:111)*	*32-bit immediate*	
CMP reg, imm32	*010110*	*(000:111)*		
AND reg, imm32	*010111*	*(000:111)*		
OR reg, imm32	*011000*	*(000:111)*		
XOR reg, imm32	*011001*	*(000:111)*		
XNOR reg, imm32	*011010*	*(000:111)*		
NOR reg, imm32	*011011*	*(000:111)*		
NAND reg, imm32	*011100*	*(000:111)*		
SLL reg, imm5	*011101*	*(000:111)*	*5-bit immediate*	
SRL reg, imm5	*011110*	*(000:111)*		
ADD reg, imm32	*011111*	*(000:111)*	*32-bit immediate*	
SUB reg, imm32	*100000*	*(000:111)*		
MUL reg, imm32	*100001*	*(000:111)*		
DIV reg, imm32	*100010*	*(000:111)*		
LD reg, imm11	*110000*	*(000:111)*	*11-bit immediate*	
ST reg, imm11	*110001*	*(000:111)*	*11-bit immediate*	

Type D Instructions

Instruction	*OP Code*	*Operand 1*
OUT imm32	*100011*	*32-bit immediate*
PUSH imm32	*100100*	
JMP imm16	*100101*	*16-bit immediate*
JZ imm16	*100110*	
JNZ imm16	*100111*	
JEQ imm16	*101000*	
JNEQ imm16	*101001*	
JG imm16	*101010*	
JL imm16	*101011*	
JGE imm16	*101100*	
JLE imm16	*101101*	

Type X Instructions

Instruction	*OP Code*
GRA	*101110*
SND	*101111*

Instruction Size

Type	*OP Code*	*Operand 1*	*Operand 2*
Type A		*3 bits*	—
Type B		*3 bits*	*3 bits*
Type C	*6 bits*	*3 bits*	*32 bits*
Type D		*32 bits*	—
Type X			

0.3 Language Rules

The VP uses a custom variant of assembly language called chasm. its rules are shown below:

- *Chasm is case-insensitve.*
- *Each line of code consists of a either:*
  - *a mnemonic*
  - *a mnemonic followed by a single operand*
  - *a mnemonic followed by an operand, a colon ”,” ,and another operand.*
- *The first operand is always a register.*
- *All numbers must begin with a 0?, where ? corresponds to the base of the number, for example, 0x53A is hex 53A, 0d123 is decimal 123, and 0b11011 is binary 11011.*
- *An instruction must have one space after the mnemonic, and one space after the colon (if applicable)*