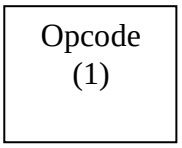
CPU Simulator Instruction Set Architecture (ISA)

The CPU simulator's instructions are multiple length instructions. Different instruction lengths vary from one byte to a maximum of 7 bytes in length. As a result, this ISA qualifies as a complex instruction set CPU (CISC) architecture as opposed to fixed-length instruction set CPU architecture, i.e. reduced instruction set CPU (RISC). The simulator instructions are composed of the following formats (numbers in brackets are lengths in bytes).

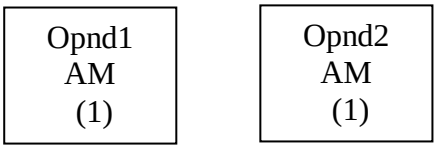
Operation code (Opcode)

The first byte of each instruction is an opcode value. Each opcode is a unique 1-byte value. Currently, the CPU simulator supports 47 opcodes (and instructions).



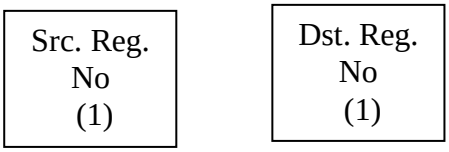
Operand addressing mode (Opnd1 AM and Opnd2 AM)

Each operand is preceded by a 1-byte value indicating the addressing mode of the operand. Currently the CPU simulator supports 8 addressing modes.



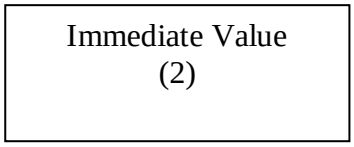
Register number (Src. Reg. No. and Dst. Reg. No.)

Each general purpose register is assigned a unique 1-byte value, starting from 0 to a maximum number (currently can vary from 8 to 64 registers). Registers are identified as source and destination registers within instructions.



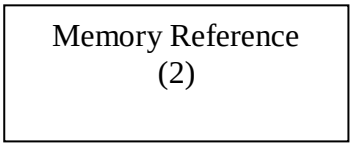
Immediate value

This is a 2-byte value indicating an immediate literal value.



Memory reference (or address)

This is a 2-byte value indicating a memory address but some instructions use this as a value which does not reference a memory location.



The following instruction formats are supported

There are 11 instruction formats. Numbers in brackets are lengths in bytes.

1)

Opcode (1)

2)

Opcode (1)	Opnd1 AM (1)	Src. Reg. No (1)

3)

Opcode (1)	Opnd1 AM (1)	Dst. Reg. No (1)

4)

Opcode (1)	Opnd1 AM (1)	Src. Reg. No (1)	Opnd2 AM (1)	Dst. Reg. No (1)

5)

Opcode (1)	Opnd1 AM (1)	Immediate Value (2)

6)

Opcode (1)	Opnd1 AM (1)	Immediate Value (2)	Opnd2 AM (1)	Dst. Reg. No (1)

7)

Opcode (1)	Opnd1 AM (1)	Src. Reg. No (1)	Opnd2 AM (1)	Memory Reference (2)

8)

Opcode (1)	Opnd1 AM (1)	Memory Reference (2)	Opnd2 AM (1)	Dst. Reg. No (1)

9)

Opcode (1)	Opnd1 AM (1)	Memory Reference (2)

10)

Opcode (1)	Opnd1 AM (1)	Immediate Value (2)	Opnd2 AM (1)	Memory Reference (2)

11)

Opcode (1)	Opnd1 AM (1)	Memory Reference (2)	Opnd2 AM (1)	Memory Reference (2)

General purpose registers

The registers are identified by an "R" followed by a two-digit decimal number indicating the register number. The register numbers can be from "00" to "07" or from "00" to "15" or from "00" to "31" or from "00" to "63" depending on the register file size selected.

e.g. "R01" refers to register 01 and "R14" refers to register 14, etc.

It is interesting to note that when the register file size of 32 or 64 are specified, this CPU architecture resembles that of a RISC architecture in this respect. This ISA is a sort of a "hybrid" ISA!

The following addressing modes are supported

Addressing mode determines how an operand is addressed. An addressing mode can be direct or indirect. Direct addressing uses the value of an operand as the target value (i.e. single level of referencing); indirect addressing uses the value of an operand to locate the actual target value (i.e. two levels of referencing). Small number of addressing modes are typical of RISC type architectures!

Value	Mode	Examples	Comments
00	Immediate	#1000	Literal value 1000
01	Register direct	R01	Register R01
02	Memory direct	1000	Memory address 1000
03	Register indirect	@R01	Memory address is in register R01
04	Memory indirect	@1000	Memory address is in memory address 1000
05	Relative address	+/-100	Jump address 100 relative to PC
06	Relative register	+/-R01	Direct register jump address relative to PC is in register R01
07	Relative indirect register	+/-@R12	Indirect register jump address relative to PC is in memory address in register R12

Instruction groups

The CPU simulator's instruction set is currently composed of the following groups of instructions

- Data transfer
- Arithmetic
- Control transfer
- Comparison
- Input/output
- Logical
- Miscellaneous

The following opcodes and instruction specifications are supported

Opcode values are in decimal and [hexadecimal] values. The 4th column shows the format(s) the instructions conform to. The numbers in round brackets represent Opnd1 and Opnd2 addressing modes respectively. The last column includes a '*' if the instruction updates the status register, SR. A word is 16 bits long.

Data transfer instructions (14)

00 [00]	MOV	Move value to register, move register to register	(1,1)4, (0,1)6	*
01 [01]	MVS	Move string in memory to memory	(3,3)4, (3,2)7, (2,3)8, (4,3)8, (2,2)11, (4,4)11	
04 [04]	LDB	Load a byte from memory to register	(3,1)4, (2,1)8, (4,1)8	*
05 [05]	LDW	Load a word from memory to register	(3,1)4, (2,1)8, (4,1)8	*
07 [07]	LDBI	Loads byte from memory into register, increments indirect source address	(3,1)4, (2,1)8, (4,1)8	*
08 [08]	LDWI	Loads word from memory into register, increments indirect source address	(3,1)4, (2,1)8, (4,1)8	*
09 [09]	TAS	Test and set	(3,1)4, (2,1)8, (4,1)8	*
10 [0A]	STB	Store a byte value to memory or from register to memory	(1,3)4, (0,3)6, (1,2)7, (1,4)7, (0,2)10, (0,4)10	
11 [0B]	STW	Store a word value to memory or from register to memory	(1,3)4, (0,3)6, (1,2)7, (1,4)7, (0,2)10, (0,4)10	
12 [0C]	STBI	Store a byte value to memory or from register to memory; increment indirect destination address	(1,3)4, (0,3)6, (1,4)7, (0,4)10	
13 [0D]	STWI	Store a word value to memory or from register to memory; increment indirect destination address	(1,3)4, (0,3)6, (1,4)7, (0,4)10	
14 [0E]	PSH	Push value or register to top of stack	(1)2, (0)5	
15 [0F]	POP	Pop a value from top of stack into register	(1)3	*
16 [10]	SWP	Swap register values	(1,1)4	

Arithmetic instructions (7)

17 [11]	ADD	Add value or register to destination register	(1,1)4, (0,1)6	*
18 [12]	SUB	Subtract value or register from destination register	(1,1)4, (0,1)6	*
19 [13]	SUBU	Subtract value in registers with unsigned result	(1,1)4, (0,1)6	*
20 [14]	MUL	Multiply value or register with	(1,1)4, (0,1)6	*

		destination register		
21 [15]	DIV	Divide value or register with destination register	(1,1)4, (0,1)6	*
22 [16]	INC	Increment register value by 1	(1)3	*
23 [17]	DEC	Decrement register value by 1	(1)3	*

Logical instructions (5)

24 [18]	AND	And value or register with destination register	(1,1)4, (0,1)6	*
25 [19]	OR	Or value or register with destination register	(1,1)4, (0,1)6	*
26 [1A]	NOT	Invert register value (bitwise)	(1,1)4, (0,1)6	*
27 [1B]	SHL	Shift register left with specified bit positions	(2,1)8	*
28 [1C]	SHR	Shift register right with specified bit positions	(2,1)8	*

Control transfer instructions (16)

29 [1D]	JMP	Unconditionally jump to memory location	(1)3, (6)3, (7)3, (2)9, (5)9	
30 [1E]	JEQ	Jump to memory location if equal	(1)3, (6)3, (7)3, (2)9, (5)9	
31 [1F]	JNE	Jump to memory location if not equal	(1)3, (6)3, (7)3, (2)9, (5)9	
32 [20]	JGT	Jump to memory location if greater than	(1)3, (6)3, (7)3, (2)9, (5)9	
33 [21]	JGE	Jump to memory location if greater than or equal	(1)3, (6)3, (7)3, (2)9, (5)9)9	
34 [22]	JLT	Jump to memory location if less than	(1)3, (6)3, (7)3, (2)9, (5)9	
35 [23]	JLE	Jump to memory location if less than or equal	(1)3, (6)3, (7)3, (2)9, (5)9	
36 [24]	JNZ	Jump if Z status flag is not set	(1)3, (6)3, (7)3, (2)9, (5)9	
37 [25]	JZR	Jump if Z status flag is set	(1)3, (6)3, (7)3, (2)9, (5)9	
38 [26]	CAL	Jump to specified subroutine memory location	(1)3, (6)3, (7)3, (2)9, (5)9	
39 [27]	LOOP	Loops if register value is greater than 0	(2, 1)8, (5,1)8	
43 [2B]	MSF	Mark stack frame	1	
44 [2C]	RET	Return from subroutine	1	
45 [2D]	IRET	Return from interrupt subroutine	1	
46 [2E]	SWI	Generate software interrupt	(2)9	
47 [2F]	HLT	Halt simulator	1	

Comparison instructions (2)

48 [30]	CMP	Compare value or register with register	(1,1)4, (0,1)6	*
49 [31]	CPS	Compare string in memory with string in memory	(3,3)4, (3,2)7, (2,3)8, (4,3)8, (2,2)11, (4,4)11	*

I/O instructions (2)

50 [32]	IN	Get characters from external input device	(2,1)8, (2,2)11	
51 [33]	OUT	Put characters to external output device	(1,2)7, (3,2)7, (0,2)10, (2,2)11, (4,2)11	

Miscellaneous instructions (1)

52 [34]	NOP	No operation	1	

Status register flags

Status register flags are used to indicate CPU status as a result of the execution of the most recent instruction. This information is often needed by some of the instructions following the last instruction that modified the status flag(s).

OV: The last instruction caused an arithmetic overflow, i.e. the result is too big to be stored in a register.
Z: The last instruction resulted in a zero result stored in a register.
N: The last instruction resulted in a negative result stored in a register.

OV	Z	N	Comment
Not set	Not set	Not set	Positive result
Not set	Set	Not set	Zero result
Not set	Not set	Set	Negative result
Set	Not set	Not set	Overflow result

All arithmetic opcodes and comparison opcodes update the status flag(s). All except the unconditional jump (JMP) instructions use the status register flag values to decide if they should jump to a specified memory location or not. The specified memory location is part of the jump instruction as an operand.

Examples of usage of CPU simulator instruction sub-set

Instruction	Description and examples of usage
Data transfer instructions	
MOV	Move data to register; move register to register e.g. **MOV #2, R01** ;moves number 2 into register R01 **MOV R01, R03** ;moves contents of register R01 into register R03
LDB	Load a byte from memory to register e.g. **LDB 1000, R02** ;loads one byte value from memory location 1000 **LDB @R00, R01** ;memory location is specified in register R00
LDW	Load a word (2 bytes) from memory to register e.g. **LDW 1000, R02** ;loads two-byte value from memory location 1000 **LDW @R00, R01** ;memory location is specified in register R00
STB	Store a byte from register to memory e.g. **STB #2, 1000** ;stores value 2 into memory location 1000 **STB R02, @R01** ;memory location is specified in register R01
STW	Store a word (2 bytes) from register to memory e.g. **STW R04, 1000** ;stores register R04 into memory location 1000 **STW R02, @2000** ;memory location is specified in memory 2000
PSH	Push data to top of hardware stack (TOS); push register to TOS e.g. **PSH #6** ;pushes number 6 on top of the stack **PSH R03** ;pushes the contents of register R03 on top of the stack
POP	Pop data from top of hardware stack to register e.g. **POP R05** ;pops contents of top of stack into register R05

TAS	Test and set - Atomic instruction (cannot be interrupted): Read memory location and store in register, store value 1 in memory location. e.g. **TAS 100, R01** ; stores value in memory location 100 in register R01 then writes a 1 in memory location 100. **TAS @R02, R03** ; stores value in memory location address of which is in register R02 in register R03 then writes a 1 in the same memory location.
Arithmetic instructions	
ADD	Add number to register; add register to register e.g. **ADD #3, R02** ;adds number 3 to contents of register R02 and stores the result in register R02. **ADD R00, R01** ;adds contents of register R00 to contents of register R01 and stores the result in register R01.
SUB	Subtract number from register; subtract register from register
MUL	Multiply number with register; multiply register with register
DIV	Divide number with register; divide register with register
INC	Increment register value by 1 e.g. **INC R08** ; increment value of register R08 by 1.
Control transfer instructions	
JMP	Jump to instruction address unconditionally e.g. **JMP 100** ;unconditionally jumps to address location 100
JLT	Jump to instruction address if less than (after last comparison) e.g. **JLT 1000** ;jumps to address location 1000 if the previous comparison instruction result indicates that operand 2 is less than operand 1. **JLT +20** ; jumps to address location 20 bytes ahead of the current address. **JLT -R04** ; jumps to address location number of bytes, found in R04, behind the current address. **JLT -@R03** ; jumps to address location number of bytes, found in

	memory address specified in R03, behind the current address.
JGT	Jump to instruction address if greater than (after last comparison)
JEQ	Jump to instruction address if equal (after last comparison) e.g. **JEQ 200** ;jumps to address location 200 if the previous comparison instruction result indicates that the two operands are equal.
JNE	Jump to instruction address if not equal (after last comparison)
LOOP	Loops until value in register is 0. Assumes that the register is initially loaded with a value. e.g. **LOOP 100, R05** ; jump to address 100 as long as the value in R05 > 0.
CAL	Jump to subroutine address e.g. To call a subroutine starting at address location 1000 use the following sequence of instructions **MSF** ; always needed just before the following instruction **CAL 1000** ; will cause a jump to address location 1000
RET	Return from subroutine e.g. The last instruction in a subroutine must always be the following instruction **RET** ;will jump to the instruction after the last CAL instruction.
SWI	Software interrupt (used to request OS services)
HLT	Halt CPU simulator
Comparison instruction	
CMP	Compare number with register; compare register with register e.g. **CMP #5, R02** ; compare number 5 with the contents of register R02 **CMP R01, R03** ; compare the contents of registers R01 and R03 Note: If R03 = R01 then the status flag **Z** will be set If R03 > R01 then non of the status flags will be set If R03 < R01 then the status flag **N** will be set
Input, output instructions	

IN	Get input data (if available) from an external IO device
OUT	Output data to an external IO device e.g. to display a string starting in memory address 120 (decimal) on console device do the following **OUT 120, 0** ;the string is in address location 120 (direct addressing) **OUT @R02, 0** ;register R02 has number 120 (indirect addressing)
Logical instructions	
AND	Perform logical (bitwise) AND operation between two operands. The result is stored in the second operand. e.g. **AND #1, R07** **AND R09, R00**
OR	Perform logical (bitwise) OR operation between two operands. The result is stored in the second operand
NOT	Perform logical (bitwise) NOT operation on first operand store the result in the second operand. e.g. **NOT #3, R01**
SHL	Shift to the left the value of register by specified number of bits. e.g. **SHL 2, R10** ; shift contents of register 10 left by 2 bit positions.
SHR	Shift to the right the value of register by specified number of bits.