Stack Operations

Stack based operations are currently not supported. This document outlines proposed stack based instructions.

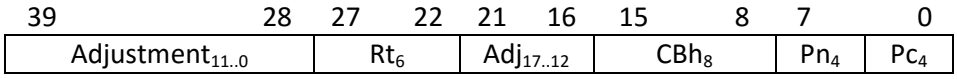
Stack based operations require en-queueing two sub-instructions in place of the stack instruction specified in program code. Logic cell requirements for the additional operations approx. 10,000 LC's.

LINK – Link Stack

Description:

The specified base pointer register Rt is pushed onto the stack, the stack pointer is loaded into register Rt and then is adjusted by the amount specified. The adjustment field of the instruction is multiplied by eight and sign extended before being applied allowing up to 128k bytes to be allocated. Note the adjustment field may not be extended with an immediate prefix. Also note the adjustment field value should be eight less than the desired value.

Instruction Format:



Clock Cycles: 4 (one memory access)

Execution Units: All ALU's / Memory

Operation:

memory[SP-8] = Rt
Rt = SP - 8
SP = SP + adjustment

PEA – Push Effective Address

Description:

An address value is calculated as the sum of the sign extended displacement and register Ra then pushed onto the stack.

Push and pop operations are unique as they enque as two instructions. This has a tendency to serialize the operation of the processor. It may improve performance in some applications to manually adjust the stack pointer, and use load / store operations instead.

Instruction Format:

3937	36 28	27 22	21 16	15 8	7 0
$\sim_3$	$Displacement_{8..0}$	$\sim_6$	Ra_6	$C9h_8$	Pn_4	Pc_4

Clock Cycles: 3 (one memory access)

Execution Units: All ALU's / Memory

Operation:

SP = SP - 8
memory[SP] = Ra + displacement

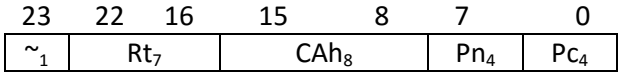
POP – Pop Register

Description:

The register is popped from the stack then the stack pointer is incremented.

Push and pop operations are unique as they enque as two instructions. This has a tendency to serialize the operation of the processor. It may improve performance in some applications to manually adjust the stack pointer, and use load / store operations instead.

Instruction Format:



Clock Cycles: 4 (one memory access)

Execution Units: All ALU's / Memory

Operation:

Rt = mem[r27]
r27 = r27 + 8

Registers Popped:

Regno (Rt_7)	Register Pushed
00 to 63	general register file
64 to 79	predicate registers #0 to #15
80 to 95	code address registers
96 to 111	segment registers
112	predicate register array
115	loop counter

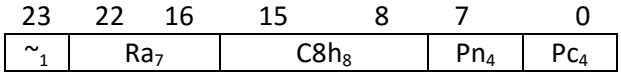
PUSH – Push Register

Description:

The stack pointer is decremented then the register is pushed onto the stack.

Push and pop operations are unique as they enque as two instructions. This has a tendency to serialize the operation of the processor. It may improve performance in some applications to manually adjust the stack pointer, and use load / store operations instead.

Instruction Format:



Clock Cycles: 3 (one memory access)

Execution Units: All ALU's / Memory

Operation:

r27 = r27 - 8
mem[r27] = Ra

Registers Pushed:

Regno (Ra_7)	Register Pushed
00 to 63	general register file
64 to 79	predicate registers #0 to #15
80 to 95	code address registers
96 to 111	segment registers
112	predicate register array
115	loop counter

UNLINK – Unlink Stack

Description:

The specified base pointer register Ra is loaded into the stack pointer, then register Ra is popped from the stack.

Instruction Format:

2322	21 16	15 8	7 0
$\sim_2$	Ra_6	CCh_8	Pn_4	Pc_4

Clock Cycles: 4 (one memory access)

Execution Units: All ALU's / Memory

Operation:

SP = Ra
Ra = memory[SP]
SP = SP + 8