The SP.4 stack computer

Paul J. Drongowski (Instructor)
Computer Engineering and Science
Case Western Reserve University
ECMP 424 Advanced Computer Architecture
Spring 1990

Introduction.

The SP.4 is a stack machine loosely based upon the Burroughs B5000 instruction set architecture (ISA.) The SP.4 uses a push-down stack for computation and subroutine calls. An SP.4 program is really a reverse Polish postfix string in which operands are pushed onto the stack and manipulated by operators. Memory references are made through a *Program Reference Table (PRT)* which maps a "virtual" operand name to its physical location in primary memory.

Memory.

SP.4 memory words are thirty-four (34) bits long. The high order two bits are "tag bits" that specify the type of data contained within the low order thirty-two bits. Primary memory has 65,536 thirty-four bit words and contains the Program Reference Table, the program itself and any data objects.

Registers.

The SP.4 has three main kinds of register objects (Table 1.) The instruction pointer, IP, selects the next program (instruction) word for execution. The Stack has an unspecified depth and operates in a last in-first out manner. The Stack contains tagged data items necessitating a 34-bit word. The PRT register contains the base address of the Program Reference Table in primary memory. The PRT register permits the easy relocation of the table.

Instructions.

Program word types (or instructions) are given in Table 2. Figure 1 provides additional detail about the bit fields within a program word by type. The

Name	Size	Purpose
IP	16 bits	Instruction pointer
Stack	? 34-bit words	Push-down stack
PRT	16 bits	Pointer to the Program Reference Table

Table 1: SP.4 register objects.

Tag	Type	Purpose
00	Operator	Perform specified stack operation
01	Literal	Push constant (literal) on stack
10	Operand call	Push operand on the stack
11	Descriptor call	Push descriptor (address) on the stack

Table 2: Program word (instruction) types.

Operator type causes a stack operation to be performed. The operation is specified in the low order five bits of the word. Stack operations are summarized in Table 3. The number of operations was kept low to reduce the overall complexity of the SP.4 implementation. Many more arithmetic and logical operations could be added to the operation set and you should feel free to do so (within reason.)

The *Literal* program word pushes a thirty-two bit constant onto the stack. The tag bits are set to zero indicating an operand, or "pure value," item. The SetTag operator can be used to set the tag bits of the top item on the stack.

The *Operand call* and *Descriptor call* program words bring operands and descriptors to the stack. These actions obtain an operand or descriptor directly, indirectly, from an array and by computation. The low order sixteen bits of the program word is an index into the Program Reference Table (Figure 2.) The entry at that position in the PRT is a tagged word and is one of four PRT word types (Table 4.)

Figure 3 depicts the format of the four call word types. Since there are four different methods of bringing two different object types to the stack,

Operation	Op code	Behavior
Reset	0	Reset stack to its empty state
Duplicate	1	Duplicate the top items
Swap	2	Interchange top two items
Load PRT	3	Pop top item (address) into the PRT register
Pop	4	Pop top item using descriptor (second item)
Add	5	Add top two item and leave result on stack
Subtract	6	Subtract top from second item; result on stack
Multiply	7	Multiply top two items; result on top of stack
Divide	8	Divide second item by top item; result on top
Branch	9	Pop and add top item to IP register
SkipOn0	10	Pop top item and skip next instruction if zero
Return	11	Return from subroutine; swap and jump
TrapReturn	12	Return from interrupt trap; jump through top
GetInput	13	Read and push value at *Input* port
PutOutput	14	Copy (write) top stack item to *Output* port
SetTag	15	Set tag of top item to instruction bits $< 6 : 5 >$

Table 3: Stack operations.

Figure 1: Program word formats.

Figure 2: Indexing into the Program Reference Table.

Tag	Type/purpose
00	32-bit operand
01	Address of operand (bits $< 15 : 0 >$)
10	Array descriptor (bits $< 31 : 16 >$=Length, bits $< 15 : 0 >$=Base)
11	Address of subroutine entry (bits $< 15 : 0 >$)

Table 4: Program Reference Table (PRT) word types.

PRT tag	Action
00	Push this word (the operand) on the stack
01	Push the operand addressed by this word
10	Push operand addressed by sum of the base and top of stack
11	Call the subroutine at this address

Table 5: Operand call summary.

Figure 3: Call word formats.

there are eight distinct cases to be examined.

Stack operations.

This section contains more detailed information about the behavior of the stack operations. The SP.4 uses a push-down stack. A stack word is thirty-four (34) bits – two tag bits plus thirty-two data (address or descriptor) bits.

The *Reset* operation sets the stack to its empty state. In other words, all four stack words are unused and are ready to accept data. The *Duplicate* operation pushes a copy of the item at the top of the stack. The top two

PRT tag	Action
00	Push this word (the operand) on the stack
01	Push this word (an address) on the stack
10	Push this word (an array descriptor)
11	Call the subroutine at this address

Table 6: Descriptor call summary.

Figure 4: Array indexing through the PRT.

elements on the stack are interchanged by the *Swap* operation. These three operations give the programmer a small, but useful set of stack manipulation instructions.

Values are ordinarily pushed onto the stack through the literal, operand call and descriptor call program words. The *Pop* operation removes and transfers the top of the stack to primary memory. The top item is the value to be written to memory and the second item on the stack is an address or descriptor which specifies the destination memory address. Both values are removed from the stack as a side-effect of the Pop operation. This scheme is compatible with the reverse Polish postfix form of an SP.4 program.

The *Add* and *Subtract* operations remove the top two items, arithmetically combine them, and leave the sum and difference (respectively) on the stack. In the case of Subtract, the topmost item is subtracted from the second item. *Multiply* and *Divide* are similar and do the obvious things.

SP.4 programs do not contain explicit physical memory addresses. This permits the relocation of data blocks in memory, including the Program Reference Table. To make subroutines and other program segments relocatable, we must also avoid the use of explicit subroutine entry and jump addresses. The *Branch* operation pops a numeric offset from the top of the stack and adds this value to the Instruction Pointer register. This permits a relative branch to a new program address and leaves the code relocatable. The *SkipOn0* operation pops the top item from the stack and will skip the execution of the next program word (most likely a Branch) if the item is zero.

Subroutines are really functions that compute a simple numeric value (i.e., an operand) or a descriptor. When a subroutine is called, the return address is pushed onto the stack. The subroutine then computes its return value on the stack. It then executes the *Return* operation which pops both the return value and return address (the first and second items, respectively.) The return address is transfered to the Instruction Pointer register and the return value is pushed, thereby making it the top-most item.

The SP.4 has two input-output ports called, obviously enough, *Input* and *Output*. The *GetInput* operation reads a data value from the Input port and pushes it onto the stack. The *PutOutput* operation writes a **copy** of the top item on the stack to the Output port. Both input and output values are thirty-four bits long (two tag bits plus thirty-two data bits as usual.)

The *Load PRT* operation pops the top item from the stack and places it in the PRT register. This operation is essential as the PRT register is not initialized at start-up. A program must also be capable of relocating the PRT if necessary.

The ability to set the tag bits of a data item is another required function. The *SetTag* operation transfers instruction bits $< 6 : 5 >$ to the two tag bits of the data item on top of the stack.

Start-up initialization.

When the SP.4 is powered-up or reset, the Instruction Pointer is set to zero. The value of the PRT register is **undefined**. The state of the stack is also **undefined**. They must be initialized explicitly by the application program (or operating system start-up code.)

Interrupts.

The SP.4 has three interrupt trap conditions. The SP.4 will trap through location 65,533 if the offset of an operand (array) or descriptor (array) call is greater than the length of the array as specified by the PRT deescriptor. The SP.4 will trap through location 65,534 on stack underflow and location 65,535 on stack overflow. Feel free to add any other exception conditions and traps that you feel are important and essential.

During a trap, the SP.4 pushes the current Instruction Pointer value on the stack and fetches the address of the trap handler from the appropriate memory location (i.e., one of locations 65,533, etc.) The *TrapReturn* operation pops the return address from the stack and transfers it to the Instruction Pointer register.