The STACK Computer

Introduction

The STACK is a model of a simple computer. It has a 0-address instruction set and supports the requirements of simple, integer-oriented programming within the obvious limitations of its memory. This document serves to provide the basic architectural description of the machine.

Hardware Description

Memory, Address, Data, Bus Information

The STACK can address 256 words of memory, each word consisting of 12 bits. The data bus can transmit 12 bits in a single transfer. The address bus can communicate 8 bits at a time. Addresses are given as 8-bit binary numbers from 00000000 to 11111111 (010 to 25510).

There is no hardware memory protection. An instruction can reference any part of memory.

Data is represented in binary sign magnitude form. The left most bit, or 12th bit, is the sign bit and is 1 for negative numbers. A 0 indicates a non-negative value. The number zero will be stored only as 000000000000. Hence, the range of values in decimal notation is -2047 to +2047. Obviously the STACK is not for number crunching.

Run-time Stack

To support procedure calls and recursion, the STACK has a run-time stack in memory. Its bottom location is at address 10100000 (16010) and its grows toward high addresses. A cpu register designated as SP holds the address of the top element of the stack. Initially, when the stack is empty, SP will hold the address 10011111 (15910). If the stack grows to the top of memory wrap around will occur, i.e., if SP holds 11111111 and is increased by 1 (via unsigned arithmetic), it will become 00000000.

Instructions that call functions modify the run-time stack and special instructions modify the value in SP. These are discussed in the section on instructions.

CPU Registers and Flags

The cpu has 16 registers that act as a stack of depth 16. The instructions implicitly use this stack for many operations. Each register that is part of the stack holds a single 12-bit word. A simple hardware circuit (TOP) indicates the location of the top word on the stack. In our simulation we will use values in the range -1 to 15 for the value maintained by TOP. The value -1 indicates that the stack is empty and is the initial value when a program starts. Two fatal interrupts STACK_OVERFLOW and STACK_UNDERFLOW are signaled when an instruction attempts to access a value on an empty stack or to place a value on a full stack.

The cpu has a number of registers and flags that are altered by instructions. We list them below in outline form.
• **Special Visible Registers**
  - IR  instruction register  12 bits
  - PC  program counter  8 bits
  - SP  stack pointer  8 bits

The SP register is used to reference the run-time stack where arguments and local variables are stored.

• **Scratch Registers**
  - T0, T1, T2  registers used with arithmetic instructions to hold operands. These are invisible to the programmer, even at the machine level. Each is 12 bits. *They need not be simulated.*

• **Flag Registers**  (1 bit each)
  - Z  set if a test instruction or an arithmetic instruction detects a zero value or result, respectively.
  - N  set if a test instruction or an arithmetic instruction detects a negative value or result, respectively.

• **Interrupt Bits**
  - X  indicates an arithmetic exception occurred. This could happen because of division by zero or because the result of an operation would not fit into 12 bits.
  - SO  indicates a STACK_OVERFLOW condition.
  - SU  indicates a STACK_UNDERFLOW condition.
  - T  indicates that a HALT instruction was processed.

---

**Processor Instruction Set**

**Notational Conventions**

- addr8 will refer to a 8-bit address.
- c8 will refer to a 8-bit unsigned constant.
- s8 will refer to a 8-bit signed constant
- s[TOP] will refer to the top item on the cpu stack.
- m[addr] will refer to the contents of memory at the specified address. addr may be replaced by a register that holds an address as in m[PC] or m[SP].

*We will use a pseudo-code to describe the effect of each instruction.*

**NOP**

There are no effects when this instruction is executed.

Encoding: 100000000000
HALT

The T interrupt is set.

Encoding: 100000000001

ADD

Addition of two stack items and their replacement by the result. The X interrupt is set if overflow occurs. The Z and N flags are cleared or set to reflect the result of the operation.

\[
\begin{align*}
T0 & \leftarrow s[\text{TOP}] \\
\text{TOP} & \leftarrow \text{TOP}-1 \\
T1 & \leftarrow s[\text{TOP}] \\
T2 & \leftarrow T0 + T1 \\
\text{s[TOP]} & \leftarrow T2
\end{align*}
\]

Encoding: 100000000010

SUB

Subtraction of two stack items and their replacement by the result. The X interrupt is set if an exception occurs. The Z and N flags are cleared or set to reflect the result of the operation.

\[
\begin{align*}
T0 & \leftarrow s[\text{TOP}] \\
\text{TOP} & \leftarrow \text{TOP}-1 \\
T1 & \leftarrow s[\text{TOP}] \\
T2 & \leftarrow T1 - T0 \quad \text{Note the order.} \\
\text{s[TOP]} & \leftarrow T2
\end{align*}
\]

Encoding: 100000000011

NEG

Negation of top item on cpu stack. The Z and N flags are cleared or set to reflect the result of the operation.

\[
\begin{align*}
T0 & \leftarrow s[\text{TOP}] \\
\text{Negate } T0 \text{ by toggling leading bit} \\
\text{s[TOP]} & \leftarrow T0
\end{align*}
\]

Encoding: 100000000100

MUL

Multiplication of two stack items and their replacement by the result. The X interrupt is set if an exception occurs. The Z and N flags are cleared or set to reflect the result of the operation.

\[
\begin{align*}
T0 & \leftarrow s[\text{TOP}] \\
\text{TOP} & \leftarrow \text{TOP}-1 \\
T1 & \leftarrow s[\text{TOP}] \\
T2 & \leftarrow T0 \ast T1 \\
\text{s[TOP]} & \leftarrow T2
\end{align*}
\]

Encoding: 100000000101
DIV

Integer division of two stack items and their replacement by the result. The X interrupt is set if an exception occurs. The Z and N flags are cleared or set to reflect the result of the operation.

\[
\begin{align*}
T0 & \leftarrow s[\text{TOP}] \\
\text{TOP} & \leftarrow \text{TOP} - 1 \\
T1 & \leftarrow s[\text{TOP}] \\
T2 & \leftarrow T1 / T0 \quad \text{(produce quotient, ignore remainder)} \\
s[\text{TOP}] & \leftarrow T2
\end{align*}
\]

Encoding: 100000000110

TEST

Set or clear Z and N according to value on top of cpu stack.

\[
\begin{align*}
T0 & \leftarrow s[\text{TOP}] \\
T1 & \leftarrow 0 \\
T2 & \leftarrow T0 - T1 \\
\text{Set Z, N based on } T2
\end{align*}
\]

Encoding: 100000000111

RET

Return from a procedure call. The programmer is responsible for assuring that SP references the saved PC value at the time RET is executed.

\[
\begin{align*}
\text{PC} & \leftarrow (\text{SP}) \quad \text{saved return address} \\
\text{SP} & \leftarrow (\text{SP} - 1) \quad \text{restore SP prior to push of args}
\end{align*}
\]

Encoding: 100000001000

ARG

Push the top of the cpu stack on the run-time stack. It is an argument to a function and this is part of the calling sequence.

\[
\begin{align*}
\text{SP} & \leftarrow \text{SP} + 1 \\
M[\text{SP}] & \leftarrow s[\text{TOP}] \\
\text{TOP} & \leftarrow \text{TOP} - 1
\end{align*}
\]

Encoding: 100000001001

RESET

The cpu stack is cleared by resetting TOP to -1.

Encoding: 100000001010

IN

A system call is made that obtains an integer from the standard input stream and places it on the top of the cpu stack. This is like a PUSH operation.

\[
\begin{align*}
\text{TOP} & \leftarrow \text{TOP} + 1 \quad \text{(check for overflow)} \\
s[\text{TOP}] & \leftarrow \text{system-provided integer}
\end{align*}
\]
OUT

A system call which causes the top value on the cpu stack to be sent to the standard output stream. This is like a POP operation.

\[
\begin{align*}
\text{stdout} & \leftarrow s[\text{TOP}] \quad \text{(check for underflow)} \\
\text{TOP} & \leftarrow \text{Top} - 1
\end{align*}
\]

Encoding: 100000001011

PUSHI

Push the signed constant \( s^8 \) onto the cpu stack with appropriate insertion of four zero bits after the sign bit so as to represent the value in 12 bits.

\[
\begin{align*}
\text{TOP} & \leftarrow \text{TOP} + 1 \quad \text{(check for overflow)} \\
s[\text{TOP}] & \leftarrow s^8 \quad \text{(extended)}
\end{align*}
\]

Encoding: 0000s8

PUSH

Push the value at address \( \text{addr}^8 \) onto the cpu stack.

\[
\begin{align*}
\text{TOP} & \leftarrow \text{TOP} + 1 \quad \text{(check for overflow)} \\
s[\text{TOP}] & \leftarrow M[\text{addr}^8]
\end{align*}
\]

Encoding: 0001addr8

POP

Store the value on the top of the cpu stack into the location with address \( \text{addr}^8 \).

\[
\begin{align*}
M[\text{addr}^8] & \leftarrow s[\text{TOP}] \quad \text{(check for underflow)} \\
\text{TOP} & \leftarrow \text{TOP} - 1
\end{align*}
\]

Encoding: 0010addr8

JMP

Unconditional branch to location \( \text{addr}^8 \)

\[
\text{PC} \leftarrow \text{addr}^8
\]

Encoding: 0011addr8

JMPZ

Conditional jump on zero.

\[
\begin{align*}
\text{if } (Z), \text{ then } & \text{PC} \leftarrow \text{addr}^8
\end{align*}
\]

Encoding: 0100addr8

JMPN

Conditional jump on negative.

\[
\begin{align*}
\text{if } (N), \text{ then } & \text{PC} \leftarrow \text{addr}^8
\end{align*}
\]

Encoding: 0101addr8
CALL

Jump to the address addr8 which is the first instruction in the subroutine after saving the SP and PC registers. The number of arguments is assumed to be on top of the cpu stack. It is used to obtain the SP value prior to push of arguments.

\[
\begin{align*}
\text{SP} & \leftarrow \text{SP} + 1 \\
\text{T0} & \leftarrow s[\text{TOP}] \quad \text{(check for underflow)} \\
\text{TOP} & \leftarrow \text{TOP} - 1 \\
\text{M[SP]} & \leftarrow \text{SP} - \text{T0} - 1 \quad \text{save value of SP before args pushed} \\
\text{SP} & \leftarrow \text{SP} + 1 \\
\text{M[SP]} & \leftarrow \text{PC} \quad \text{save PC for return} \\
\text{PC} & \leftarrow \text{addr8} \quad \text{jump to subroutine}
\end{align*}
\]

Encoding: 0110addr8

INC-SP

Increase SP by the value of the unsigned constant c8. This can cause wrap around in memory. Normally this value will not be large.

\[
\text{SP} \leftarrow \text{SP} + c8
\]

Encoding: 1100c8

DEC-SP

Decrease SP by the value of the unsigned constant c8. This can cause wrap around in memory. Normally this value will not be large.

\[
\text{SP} \leftarrow \text{SP} - c8
\]

Encoding: 1101c8

PUSH-REL-SP

Push item in the memory location with address SP + s8 onto the cpu stack, where s8 is a signed 8-bit value.

\[
\begin{align*}
\text{TOP} & \leftarrow \text{TOP} + 1 \quad \text{(check for overflow)} \\
 s[\text{TOP}] & \leftarrow M[\text{SP} + s8]
\end{align*}
\]

Encoding: 1110s8

POP-REL-SP

Store the top of the cpu stack in memory at the location that has address SP + s8 where s8 is a signed 8-bit value.

\[
\begin{align*}
 M[\text{SP} + s8] & \leftarrow s[\text{TOP}] \quad \text{(check for underflow)} \\
 \text{TOP} & \leftarrow \text{TOP} - 1
\end{align*}
\]

Encoding: 1111s8
Calling Sequence

If one wishes to call the subroutine foo(arg_0, arg_1, ..., arg_K), the following instructions will be executed (we give the symbolic versions):

\[
\begin{align*}
&\text{<get arg}_k\text{ onto top of cpu stack>} \\
&\text{ARG} \\
&\text{<get arg}_{K-1}\text{ onto cpu stack>} \\
&\text{ARG} \\
&\ldots \\
&\text{<get arg}_0\text{ onto cpu stack>} \\
&\text{ARG} \\
&PUSHI K+1 \quad \text{<puts # of args onto cpu stack>} \\
&\text{CALL foo}
\end{align*}
\]

Local variables live on the stack with space made available by the INC-SP instruction.

The return value from a function must be placed on top of the cpu-stack prior to the execution of a return (RET) instruction.

Run-time Stack After A Call

Assume that we have just called a function with two arguments. The PC will be set to the address of the first instruction of the function.

\[
\begin{array}{|l|}
\hline
\text{<locals of callee here>} \\
\hline
\text{return address, saved PC} \\
\hline
\text{saved former SP} \\
\hline
\text{first arg} \\
\hline
\text{second arg} \\
\hline
\text{locals of calling routine, if any} \\
\hline
\end{array}
\]

new SP

former SP
## Table for Instructions and their Encoding

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>Binary Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>100000000000</td>
</tr>
<tr>
<td>HALT</td>
<td>100000000001</td>
</tr>
<tr>
<td>ADD</td>
<td>100000000010</td>
</tr>
<tr>
<td>SUB</td>
<td>100000000011</td>
</tr>
<tr>
<td>NEG</td>
<td>100000000100</td>
</tr>
<tr>
<td>MUL</td>
<td>100000000101</td>
</tr>
<tr>
<td>DIV</td>
<td>100000000110</td>
</tr>
<tr>
<td>TEST</td>
<td>100000000111</td>
</tr>
<tr>
<td>RET</td>
<td>100000001000</td>
</tr>
<tr>
<td>ARG</td>
<td>100000001001</td>
</tr>
<tr>
<td>RESET</td>
<td>100000001010</td>
</tr>
<tr>
<td>IN</td>
<td>100000001011</td>
</tr>
<tr>
<td>OUT</td>
<td>100000001100</td>
</tr>
<tr>
<td>PUSHI</td>
<td>0000s8</td>
</tr>
<tr>
<td>PUSH</td>
<td>0001addr8</td>
</tr>
<tr>
<td>POP</td>
<td>0010addr8</td>
</tr>
<tr>
<td>JMP</td>
<td>0011addr8</td>
</tr>
<tr>
<td>JMPZ</td>
<td>0100addr8</td>
</tr>
<tr>
<td>JMPN</td>
<td>0101addr8</td>
</tr>
<tr>
<td>CALL</td>
<td>0110addr8</td>
</tr>
<tr>
<td>INC-SP</td>
<td>1100c8</td>
</tr>
<tr>
<td>DEC-SP</td>
<td>1101c8</td>
</tr>
<tr>
<td>PUSH-REL-SP</td>
<td>1110s8</td>
</tr>
<tr>
<td>POP-REL-SP</td>
<td>1111s8</td>
</tr>
</tbody>
</table>