The CPRL Virtual Machine

See Appendix E of the textbook for additional details on the definition of CVM.

1

CVM
(**C**PRL **V**irtual **M**achine)

- CVM
  - is a hypothetical computer designed to simplify the code generation phase of a compiler for CPRL
  - uses a stack architecture; i.e., most instructions either expect values on the stack, place results on the stack, or both
  - has 4 internal or special-purpose registers, but no general-purpose registers
- Memory is organized into 8-bit bytes. Each byte is directly addressable.
- A word is a logical grouping of 4 consecutive bytes in memory. The address of a word is the address of its first (low) byte.

2

Representing Primitive Types

- Boolean values are represented in a single byte.
  - 0 means false.
  - Any nonzero value is interpreted as true.
- Character values (Unicode) use 2 bytes.
  - Unicode Basic Multilingual Plane (Plane 0)
  - Code points range from U+0000 to U+FFFF
- Integer values use a word.
  - 32-bit 2's complement

3

CVM Instructions

- Each CVM instruction operation code (opcode) occupies one byte of memory.
- Some instructions take one or two arguments, which are always located in the words immediately following the instruction in memory.
- Depending on the opcode, an argument can be
  - a single byte
  - two bytes (e.g., for a char)
  - four bytes (e.g. for an integer or a memory address)
  - multiple bytes (e.g., for a string literal)

4

CVM Instructions
(continued)

- Most instructions get their operands from the stack.
- In general, the operands are removed from the stack whenever the instruction is executed, and any results are left on the top of the stack.

5

Examples: CVM Instructions

- `ADD`: Remove two integers from the top of the stack and push their sum back onto the stack.
- `INC`: Add 1 to the integer at the top of the stack.
- `LOADW` (load word): Load/push a word (four consecutive bytes) onto the stack. The address of the first byte of the word is obtained by popping it off the top of the stack.
- CMP (compare): Remove two integers from the top of the stack and compare them. Push a byte representing -1, 0, or 1 back onto the stack depending on whether the first integer is less than, equal to, or greater than the first integer, respectively.

6

Registers

Four 32-bit internal registers (no general-purpose registers)

- PC (program counter; a.k.a. instruction pointer) - holds the address of the next instruction to be executed.
- SP (stack pointer) - holds the address of the top of the stack. The stack grows from low-numbered memory addresses to high-numbered memory addresses.
- SB (stack base) – holds the address of the bottom of the stack. When a program is loaded, SB is initialized to the address of the first free byte in memory.
- BP (base pointer) - holds the base address of the subprogram currently being executed.

7

Relative Addressing Using the SB and BP Registers

- Variables declared at program scope are addressed relative to the SB register.
- Variables declared at subprogram scope are addressed relative to the BP register.
- Example: If SB has the value 112, and program scoped variable x has the relative address 8, then the actual address of x is [SB] + relAddr(x) or 120.
- When preparing for code generation, the compiler needs to determine the relative address of every variable.
- For programs that don't have subprograms, both SB and BP will point to the same memory location.

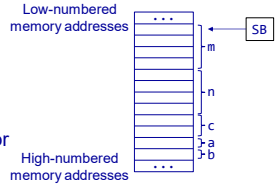
8

Relative Addressing Example

- Suppose a program contains the following declarations:

```
var m, n : Integer;
var c : Char;
var a, b : Boolean;
```

- The relative addresses of the variables are as follows:
  - m has relative address 0
  - n has relative address 4
  - c has relative address 8
  - a has relative address 10
  - b has relative address 11
- The total variable length for the program is 12.



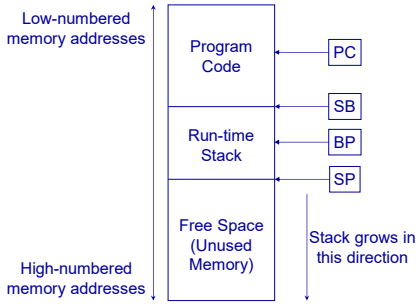
9

Loading a Program

- The object code is loaded into the beginning of memory
- Register PC is initialized to 0, the address of the first instruction.
- Registers SB and BP are initialized to the address following the last instruction.
- Register SP is initialized to BP - 1.
- The first instruction usually has the form `program n`. When executed it allocates n bytes on the top of the stack for global variables.

10

Program Loaded in Memory
(after several instructions have been executed)



11

Opcodes LDGADDR and LDLADDR

- LDGADDR n
  - load global address for variable at offset n
  - pushes SB + n onto the stack
  - used for variables declared at program scope
- LDLADDR n
  - load local address for variable at offset n
  - pushes BP + n onto the stack
  - used for variables declared at subprogram scope

12

CPRL Example

```
var   m, n : Integer;
var   c : Char;
const five := 5;

begin

   m := 7;
   n := five*m;
   c := 'X';
   writeln "n = ", n;
   writeln "c = ", c;

end.
```


13

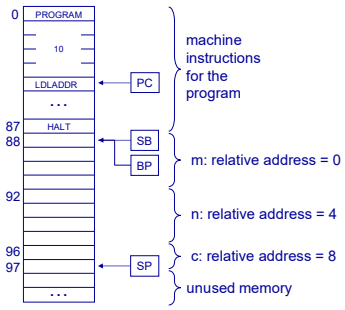
CPRL Example Disassembled

```
 0: PROGRAM 10
 5: LDGADDR 0
10: LDCINT 7
15: STOREW
16: LDGADDR 4
21: LDCINT 5
26: LDGADDR 0
31: LOADW
32: MUL
33: STOREW
34: LDGADDR 8
39: LDCCH 'X'
42: STORE2B
43: LDCSTR  "n = "
56: PUTSTR
57: LDGADDR 4
62: LOADW
63: PUTINT
64: PUTEOL
65: LDCSTR  "c = "
78: PUTSTR
79: LDGADDR 8
84: LOAD2B
85: PUTCH
86: PUTEOL
87: HALT
```

```
m: relative address = 0, absolute address = 88
n: relative address = 4, absolute address = 92
c: relative address = 8, absolute address = 96
```


14

CPRL Example in Memory
(after execution of first instruction)



15

Using the Stack to Hold Temporary Values

- The part of memory below the CVM instructions and global variables is used as a run-time stack that holds subprogram activation records (see Chapter 13) and temporary, intermediate values.
- As machine instructions are executed, the stack grows and shrinks.
- The run-time stack is empty at both the start and end of the each CPRL statement in the main program.

16

Example: Using the Stack to Hold Temporary Values

- Assume
  - register `SB` has the value `100`
  - integer variable `x` has relative address `0`
  - integer variable `y` has value `5` and relative address `4`
  - integer variable `z` has value `13` and relative address `8`
- The CPRL assignment statement

  `x := 2*y + z;`

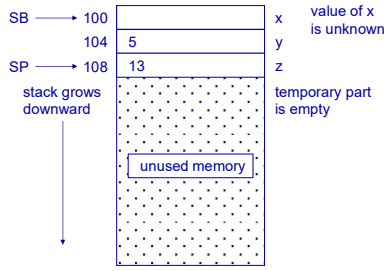
  will compile to the following CVM instructions:

```
LDGADDR 0          LDGADDR 8
LDCINT 2           LOADW
LDGADDR 4          ADD
LOADW              STOREW
MUL
```


17

Example: Using the Stack to Hold Temporary Values (continued)

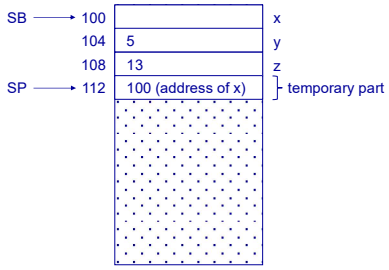
Stack is empty at the start of the CPRL statement.



18

Example: Using the Stack to Hold Temporary Values (continued)

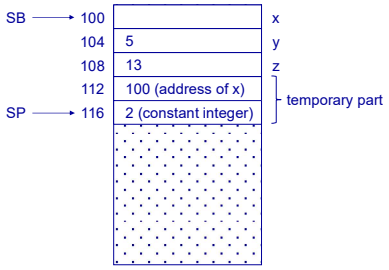
After execution of `LDGADDR 0`



19

Example: Using the Stack to Hold Temporary Values (continued)

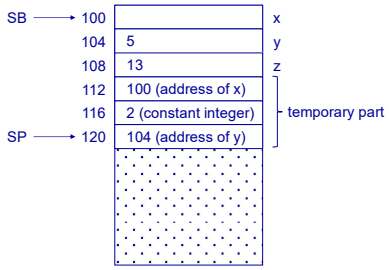
After execution of `LDCINT 2`



20

Example: Using the Stack to Hold Temporary Values (continued)

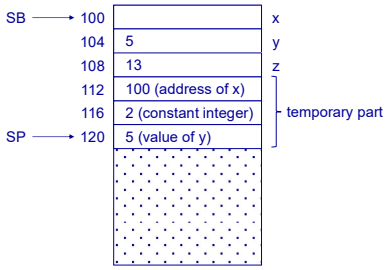
After execution of `LDGADDR 4`



21

Example: Using the Stack to Hold Temporary Values (continued)

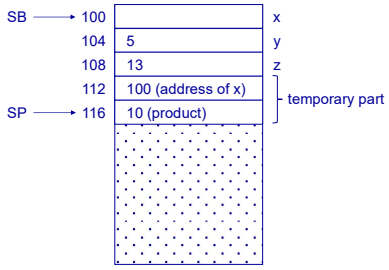
After execution of `LOADW`



22

Example: Using the Stack to Hold Temporary Values (continued)

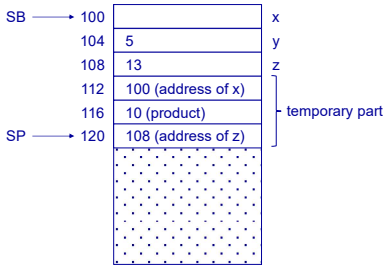
After execution of `MUL`



23

Example: Using the Stack to Hold Temporary Values (continued)

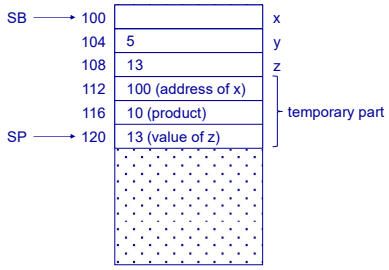
After execution of `LDGADDR 8`



24

Example: Using the Stack to Hold Temporary Values (continued)

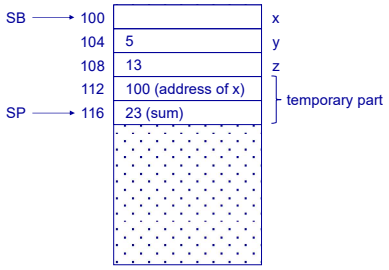
After execution of LOADW



25

Example: Using the Stack to Hold Temporary Values (continued)

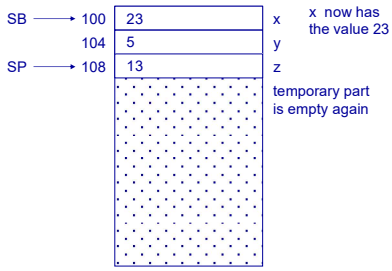
After execution of ADD



26

Example: Using the Stack to Hold Temporary Values (continued)

After execution of STOREW



27