C−− Language V2.3

Tsan-sheng Hsu

tshsu@iis.sinica.edu.tw

`http://www.iis.sinica.edu.tw/~tshsu`

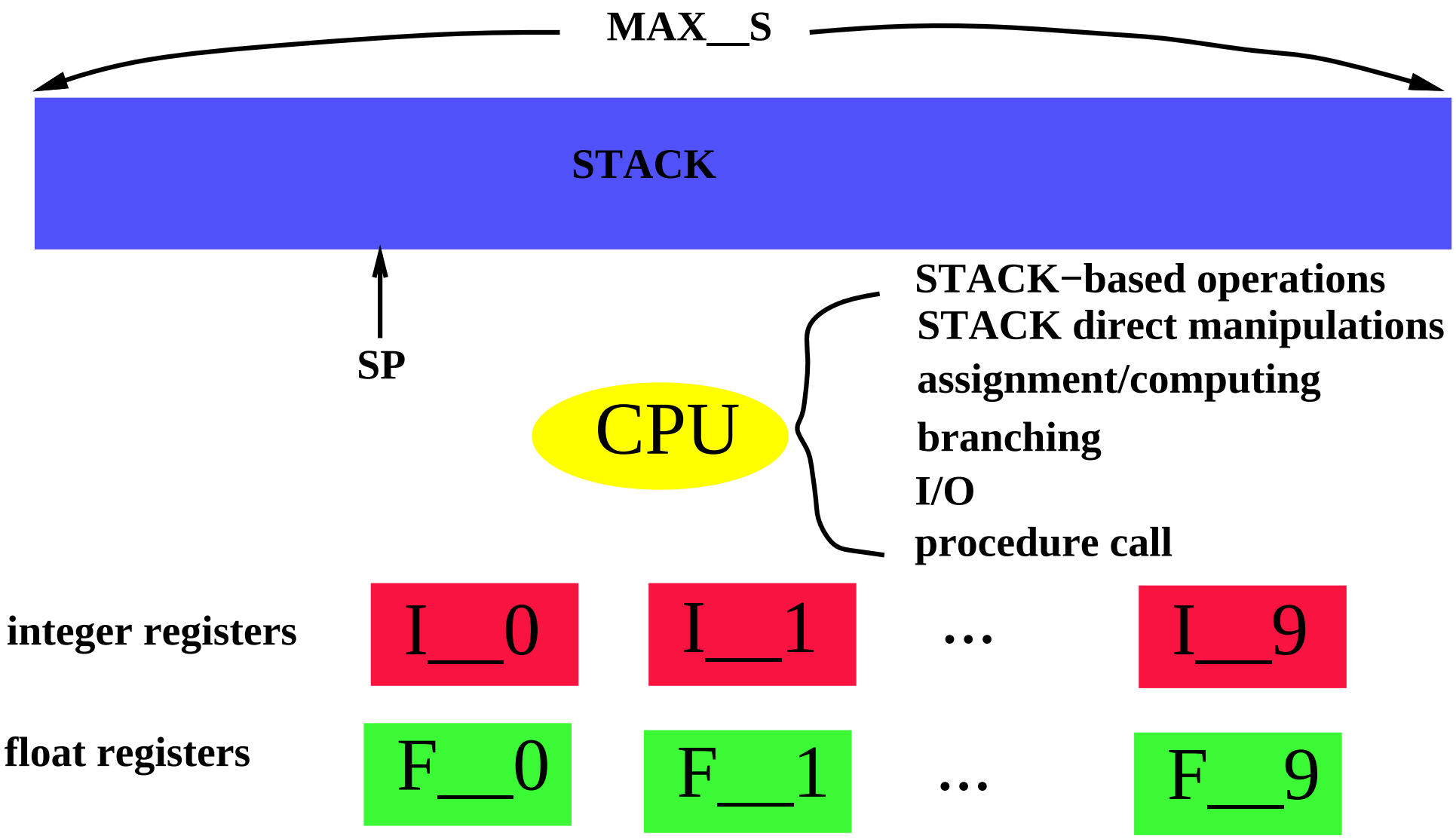
Definition

- The C−− language is a subset of the standard C language.
- Its purpose is to act like a universal intermediate language.
- C−− is a STACK based language.
- A C−− program consists of the following parts.
  - #define MAX_S maximum_stack_size
    - *Declare the size of the STACK.*
    - *Each STACK element can hold an integer or a float. That is, we assume the sizes of an integer and a float are the same.*
  - #include "cmm.c"
    - *this line is required and the file "cmm.c" contains system defined functions and variables.*
  - procedure_1()
  - procedure_2()
  - ...
  - procedure_n()

Procedure definition

- **Each procedure_i is a standard C procedure without parameters and variable declarations.**
  - **procedure_i()**
  - **{**
  - **...**
  - **}**
- **The last procedure procedure_n must be *main*.**
  - **The first statement of *main* is INIT__S();**
- **Inside each procedure, the following rules are enforced.**
  - **No variable declaration is allowed.**
  - **Constants are a single character constant, or zero, positive or negative integers / floats.**
  - **Ten global $sizeof(int)$-byte integer registers.**
    - ***They are I__0, . . . , I__9.***
    - ***These variables are called* integer registers, *or I_register.***
  - **Ten global $sizeof(float)$-byte floating point registers.**
    - ***They are F__0, . . . , F__9.***
    - ***These variables are called* float registers, *or F_register.***

Virtual machine architecture



Statements

- **Each line contains exactly one statement.**
- **Type of statements.**
  - **Null statement — a blank line containing only white spaces (tab and blank).**
  - **Comments of the form**
    - ▷ $/*\cdots*/$
  - **Stack-based operations.**
  - **Stack direct manipulations.**
  - **Assignment statements.**
  - **A C label of the form**
    - ▷ *label:*
  - **Jump statements.**
  - **I/O statements.**
  - **Type conversion statements.**
  - **Procedure call statements.**
    - ▷ $procedure\_i();$
    - ▷ $procedure\_i$ *can call* $procedure\_j$ *if* $i \geq j$.

Stack-based operations

- **INIT__S();**
  - *Initialize the stack; used only as the first statement of the procedure* **main.**
- **I_register = TOP__S();**
  - *returns the current stack pointer.*
  - *Initial value is 0.*
- **SETSP__S(I_register | I_constant);**
  - *set new stack pointer to be current stack pointer + (I_register | I_constant)*
- **ASETSP__S(I_register | I_constant);**
  - *set new stack pointer to be (I_register | I_constant)*
- **PUSH__S(I_register | I_constant);**
  FPUSH__S(F_register | F_constant);
  - *push an item into the stack;*
- **I_register = POP__S();**
  F_register = FPOP__S();
  - *pop an item from the stack;*

Stack direct manipulations

- **I_register = VAL__S(I_register | I_constant);**
  F_register = FVAL__S(I_register | I_constant);
  - *returns the value at STACK[stack pointer + (I_register | I_constant)]*
- **I_register = AVAL__S(I_register | I_constant);**
  F_register = FAVAL__S(I_register | I_constant);
  - *returns the value at STACK[(I_register | I_constant)]*
- **SSET__S(I_register_A | I_constant_A, I_register | I_constant);**
  FSSET__S(I_register_A | I_constant_A, F_register | F_constant);
  - *store the value (I_register | I_constant) (respectively, (F_register | F_constant) to STACK[stack pointer + (I_register_A | I_constant_A)];*
- **ASSET__S(I_register_A | I_constant_A, I_register | I_constant);**
  FASSET__S(I_register_A | I_constant_A, F_register | F_constant);
  - *store the value (I_register | I_constant) (respectively, (F_register | F_constant) to STACK[(I_register_A | I_constant_A)];*

Assignment statements

- **register = (register | constant);**
  - ▷ *No type conversion.*
- **register = (register | constant)** $(+|-|*|/|\%)$ **(register | constant);**
  - ▷ *No type conversion.*
- **left shift or right shift**
  - **Only for integers.**
  - **l_register** $<<=$ **(l_register | l_constant);**
  - **l_register** $>>=$ **(l_register | l_constant);**
- **l_register = (l_register | l_constant)** $(\&, \hat{}, |)$ **(l_register | l_constant);**
  - **Only for integers.**
  - **bit-wise AND, XOR and OR.**

Jump statements

- **Conditional jump**
  - **if '(' (I_register | I_constant)** $(> \mid < \mid == \mid >= \mid <=)$ **0 ')' goto label;**
  - **if '(' (F_register | F_constant)** $(> \mid < \mid == \mid >= \mid <=)$ **0.0 ')' goto label;**
- **Unconditional jump**
  - **goto label;**

I/O statements

- **Read an integer / a float into a register**
  - **scanf("%d",&I_register);**
  - **scanf("%f",&F_register);**
- **Print an integer / a float that is stored in a register**
  - **printf("%d",I_register);**
  - **printf("%f",F_register);**
- **Print an ASCII character stored in an integer register**
  - **printf("%c",I_register);**
- **Print a string**
  - **printf("string");**
- **Print a newline**
  - **printf("\n");**

Type conversion statements

- **Convert a float to an integer**
  - **I_register = (int) (F_register | F_constant);**
- **Convert an integer to a float**
  - **F_register = (float) (I_register | I_constant);**

A Sample C−− program

```
#define MAX__S 10000
#include "cmm.c"
factorial()
{
        I__2 = 1;
loop:
        I__3 = POP__S();
        if(I__3 == 0) goto ends;
        I__2 = I__2 * I__3;
        I__3 = I__3 - 1;
        PUSH__S(I__3);
        goto loop;
ends:
        PUSH__S(I__2);
}
main()
{
        INIT__S();
```

```
        I__0 = 1;
        scanf("%d",&I__1);
        if(I__1 <= 0) goto done;
        PUSH__S(I__1);
/* compute factorial */
        factorial();
compute:
        I__1 = POP__S();
        I__1 = I__1 - 2;
        if(I__1 <= 0) goto done;
        PUSH__S(I__1);
        I__0 = I__0 * I__1;
        goto compute;
done:
        printf("%d",I__0);
        printf("\n");
}
```

The file “cmm.c”

```
/* C--, version 2.3, June 2, 2006 */
#include <stdio.h>
/* stack element type */
typedef int ITYPE;
typedef float FTYPE;
typedef union u_type { ITYPE ival; FTYPE fval;} S__TYPE;
S__TYPE *STACK__S; /* stack */
ITYPE SP__S; /* stack pointer */
/* integer registers */
ITYPE I__0,I__1,I__2,I__3,I__4,I__5,I__6,I__7,I__8,I__9;
FTYPE F__0,F__1,F__2,F__3,F__4,F__5,F__6,F__7,F__8,F__9;

/* initial stack */
void INIT__S(void)
{
  STACK__S = (S__TYPE *) malloc(sizeof(S__TYPE) * (MAX__S+1));
  SP__S = 0;
}
```

```
/* return top of stack pointer */
ITYPE CURRENT__SP(void)
{
  return(SP__S);
}

/* returns the int value at stack pointer + i */
ITYPE VAL__S(i)
ITYPE i;
{
  return(STACK__S[SP__S+i].ival);
}

/* returns the int value at STACK[i] */
ITYPE AVAL__S(i)
ITYPE i;
{
  return(STACK__S[i].ival);
}
```

```
/* returns the float value at stack pointer + i */
FTYPE FVAL__S(i)
ITYPE i;
{
  return(STACK__S[SP__S+i].fval);
}

/* returns the float value at STACK[i] */
FTYPE FAVAL__S(i)
ITYPE i;
{
  return(STACK__S[i].fval);
}

/* set new stack pointer to be current stack pointer $+ i$ */
void SETSP__S(i)
ITYPE i;
{
  SP__S += i;
```

```
}

/* set new stack pointer to be  $i$ */
void ASETSP__S(i)
ITYPE i;
{
  SP__S = i;
}

/* set the int value at stack pointer $+ i$ to the int value $k$ */
void SSET__S(i,k)
ITYPE i;
ITYPE k;
{
  STACK__S[SP__S+i].ival = k;
}

/* set the int value at STACK[i] to the int value $k$ */
void ASSET__S(i,k)
ITYPE i;
```

```
ITYPE k;
{
  STACK__S[i].ival = k;
}

/* set the int value at stack pointer $+ i$ to the int value $k$ */
void FSSET__S(i,k)
ITYPE i;
FTYPE k;
{
  STACK__S[SP__S+i].fval = k;
}

/* set the int value at STACK[i] to the int value $k$ */
void FASSET__S(i,k)
ITYPE i;
FTYPE k;
{
  STACK__S[i].fval = k;
}
```

```
/* push int value k into stack */
void PUSH__S(k)
ITYPE k;
{
  SP__S += 1;
  STACK__S[SP__S].ival = k;
}

/* push float value k into stack */
void FPUSH__S(k)
FTYPE k;
{
  SP__S += 1;
  STACK__S[SP__S].fval = k;
}

/* pop int value from stack */
ITYPE POP__S(void)
{
```

```
  return(STACK__S[SP__S--].ival);
}

/* pop float value from stack */
FTYPE FPOP__S(void)
{
  return(STACK__S[SP__S--].fval);
}
```