Final Project

Peng-Sheng Chen

Department of Computer Science
National Chung Cheng University
2018

Example

- Simple expression calculator: evaluate expressions containing +, -, * and variable assignments.
- Reference from http://www.antlr.org/wiki/display/ANTLR3/Expression+evaluator

```
$
x=1
y=2
3*(x+y)  <EOF>
9
$
```

Tokens in Lexer

```
ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t')+ {skip();} ;
```

```
$
x=1
y=2
3*(x+y)  <EOF>
9
$
```

Grammar in Parser (1)

```
prog:    stat+ ;

stat:    expr NEWLINE
     |   ID '=' expr NEWLINE
     |   NEWLINE
     ;

expr
     :   multExpr
         (   '+' multExpr
         |   '-' multExpr
         )*
     ;
```

Grammar in Parser (2)

```
multExpr
     :    atom ('*' atom)*
     ;

atom
     :    INT
     |    ID
     |    '(' expr ')'
     ;
```

Actions in Parser (1)

```
grammar Expr;

@header {
import java.util.HashMap;
}

@members {
/** Map variable name to Integer object holding value */
HashMap memory = new HashMap();
}

prog:   stat+ ;
```

The @members section is where you place instance variables and methods that will be placed and used in the generated parser

Actions in Parser (2)

```
prog:    stat+ ;

stat:    expr NEWLINE
         { System.out.println($expr.value); }
     |   ID '=' expr NEWLINE
         { memory.put($ID.text,
                      new Integer($expr.value));}
     |   NEWLINE
     ;
```

Actions in Parser (3)

```
expr returns [int value]
    :   multExpr
        (   '+' multExpr
        |   '-' multExpr
        )*
    ;
```

Actions in Parser (4)

```
expr returns [int value]
    :    a=multExpr {value = $a.value;}
         (    '+' b=multExpr {value += $b.value;}
         |    '-' c=multExpr {value -= $c.value;}
         )*
    ;
```

Actions in Parser (5)

```
multExpr returns [int value]
    :    a=atom {value = $a.value;}
         ('*' b=atom {value *= $b.value;})*
    ;
```

Actions in Parser (6)

```
atom returns [int value]
    : INT {value = Integer.parseInt($INT.text);}
    | ID
      {
      Integer v = (Integer)memory.get($ID.text);
      if (v != null)
         value = v.intValue();
      else
         System.err.println("undefined var: "+$ID.text);
      }
    | '(' expr ')' {value = $expr.value;}
    ;
```

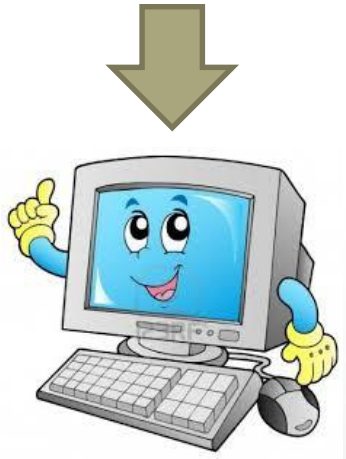
Test Class

```
import org.antlr.runtime.*;

public class TestExpr {
    public static void main(String[] args)
    {
        CharStream input = new ANTLRFileStream(args[0]);
        ExprLexer lexer = new ExprLexer(input);
        CommonTokenStream tokens = new
CommonTokenStream(lexer);

        ExprParser parser = new ExprParser(tokens);
        parser.prog();
    }
}
```

Construct Your Compiler

```
int main()
{
    int a;
    int b;

    a = 1;
    b = a + 2;
    printf("%d", b);
}
```

X86 assembly code



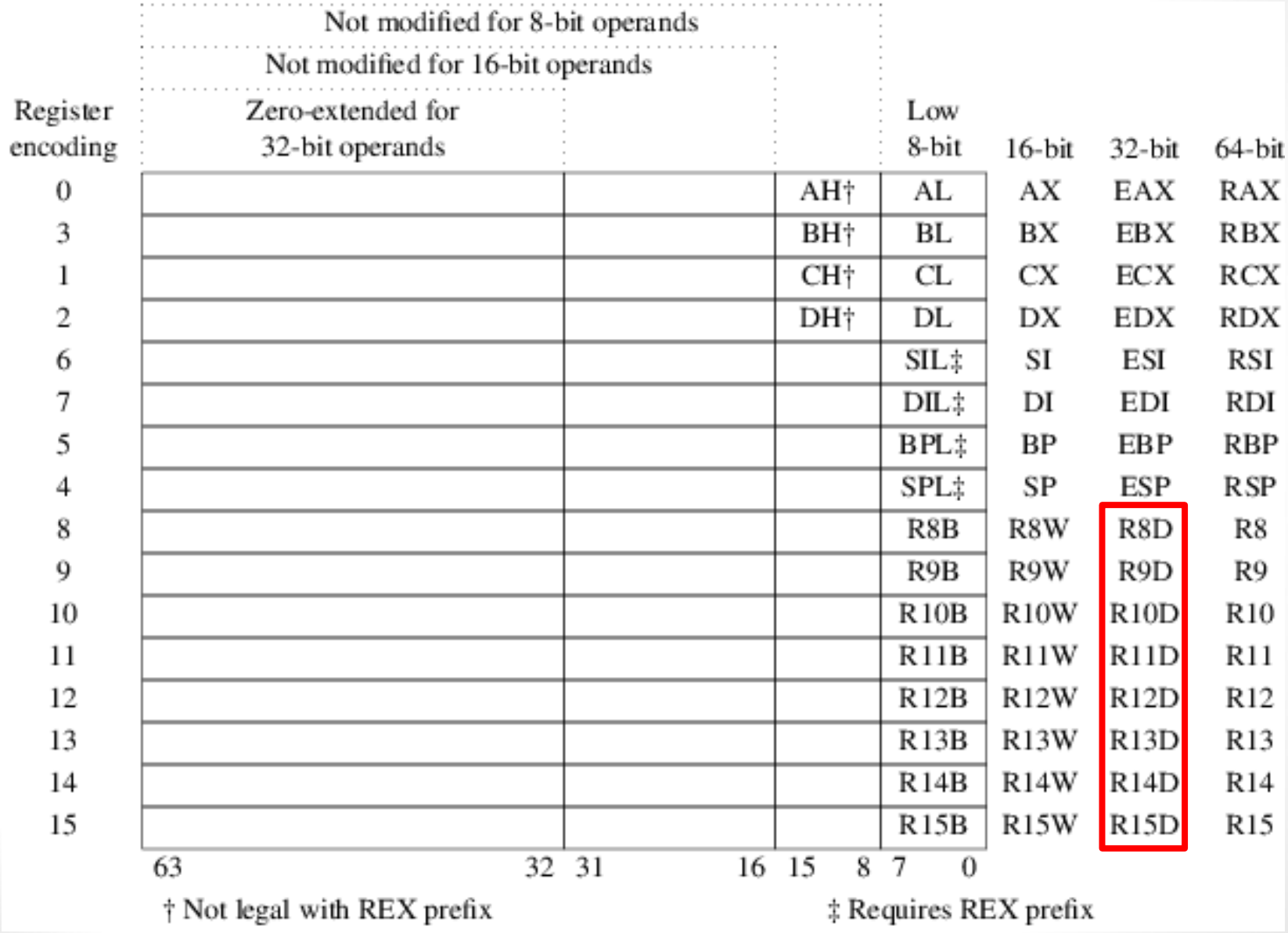
- mcore8.cs.ccu.edu.tw
- linux.cs.ccu.edu.tw

Target Platform

- 64-bit Linux (x86)
  - mcore8.cs.ccu.edu.tw
  - linux.cs.ccu.edu.tw

- Registers
  - ip => instruction pointer (program counter)
  - sp => stack pointer
  - bp => base pointer (frame pointer)

X86 Registers



Assembly Format (1)

- Intel/Microsoft format
  - Opcode dst, src
- AT&T format
  - Opcode src, dst

Assembly Format (2)

- AT&T Syntax
  - GCC uses this syntax
  - Register naming
    - Prefix with % as in %eax
  - Source and destination order
    - Reversed

mov eax, ebx

is written as

movl %ebx, %eax

  - Operand size
    - Explicit using b, w, l, q for 8bit, 16bit, 32bit, and 64bit operands

如何判別？

- 找特殊的指令 Ex: mov eax, $10

Immediate一定不會是dst

- Ex: mov $10, eax (AT&T format)

主要的範例與說明，以**AT&T format**為主

IA32 Calling Convention (1)

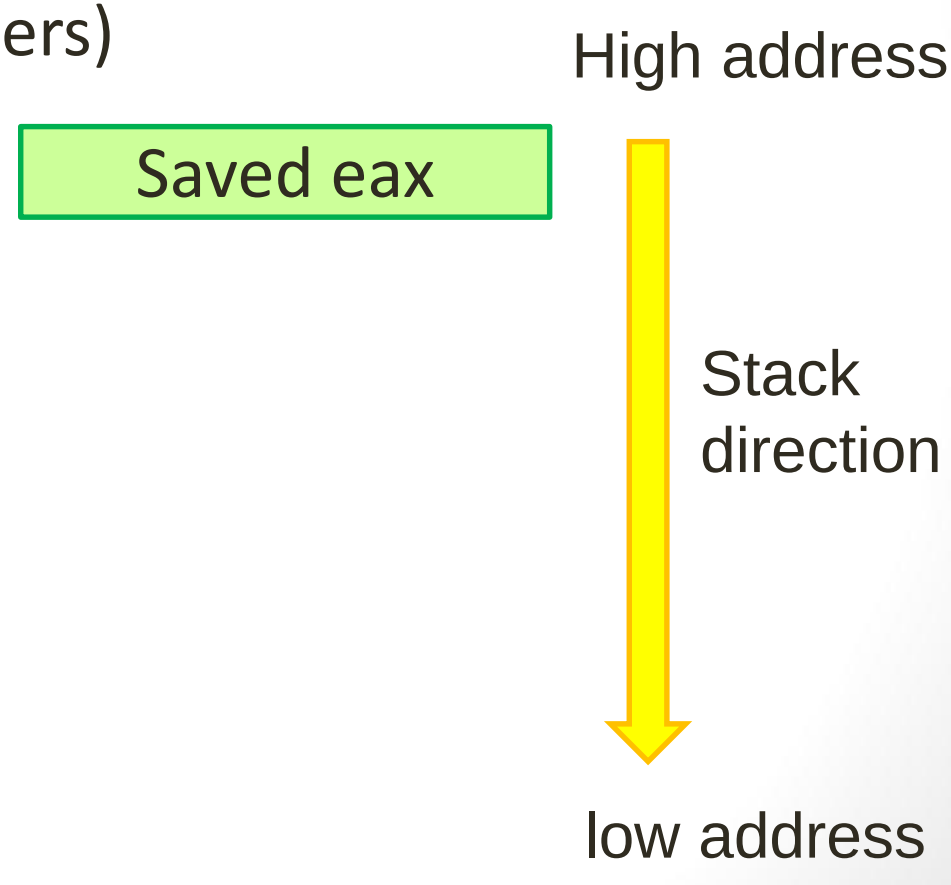
```
int Bar(int a, int b, int c);

void Foo(void)
{
   /* Some stuff here */
   Bar(42, 21, 84);
}

int Bar(int a, int b, int c)
{
   int loc;

   /* Some stuff here */
   return 1337;
}
```

IA32 Calling Convention (2)

- The return value will be stored in the eax register so the caller first have to push its current value.

  (Caller saved registers)

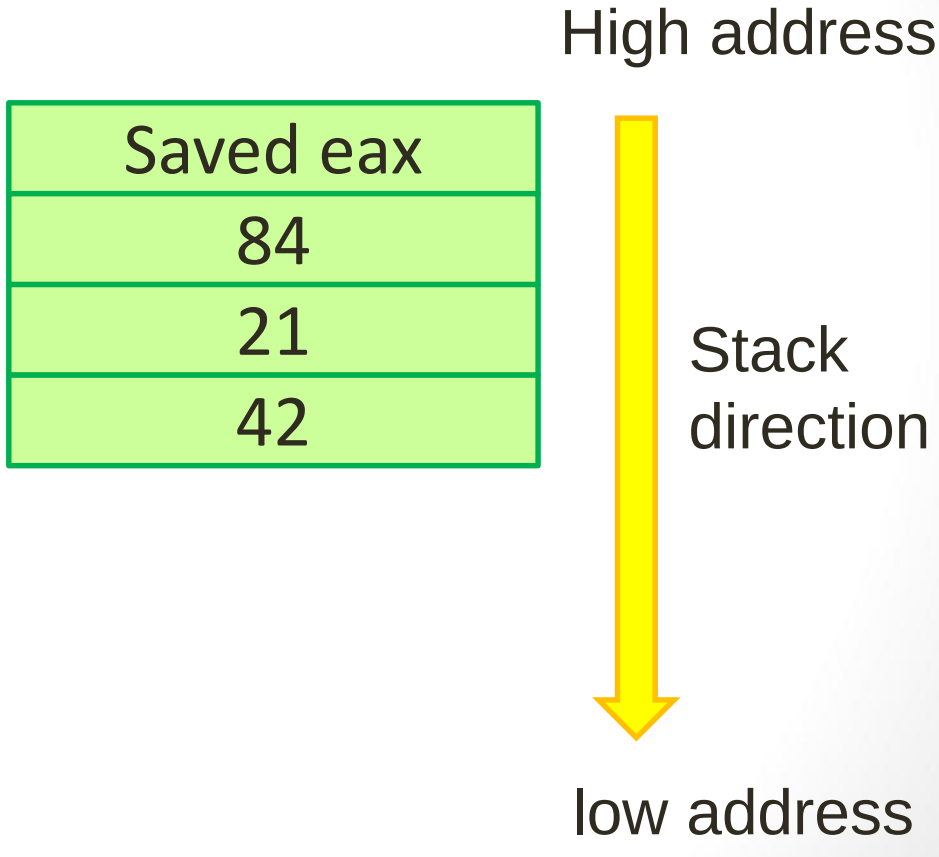
push eax



IA32 Calling Convention (3)

- The caller pushes parameters in reverse order.

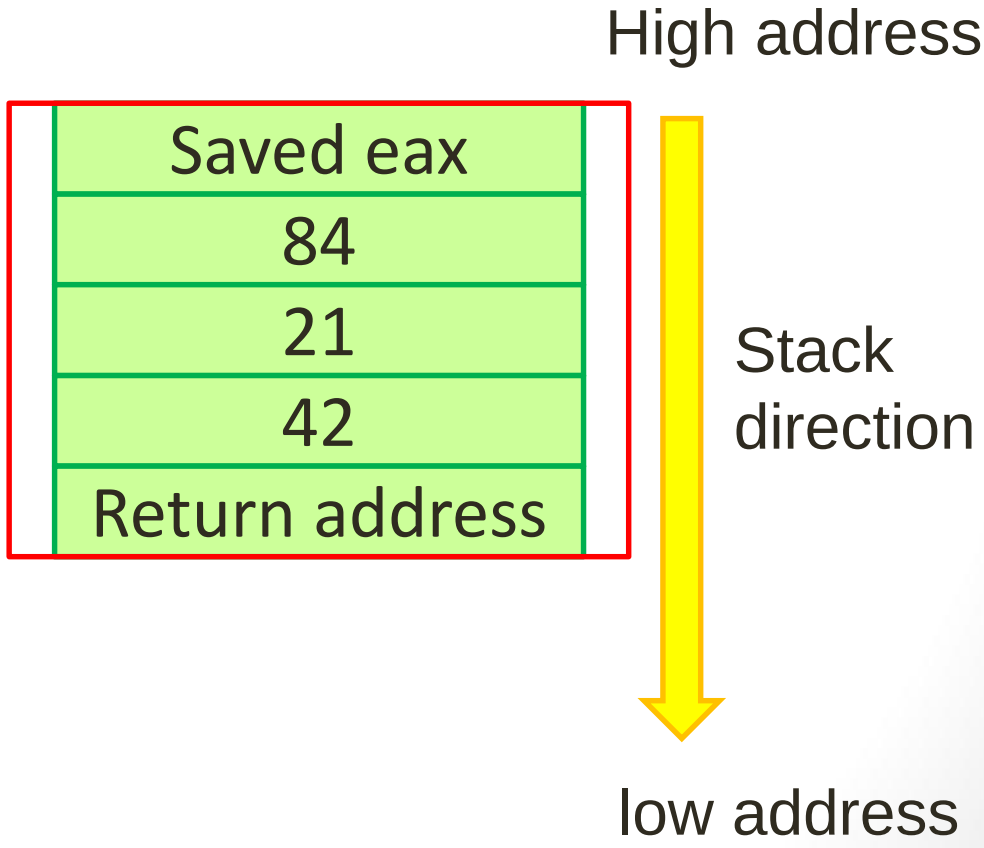
```
push  84
push  21
push  42
```



IA32 Calling Convention (4)

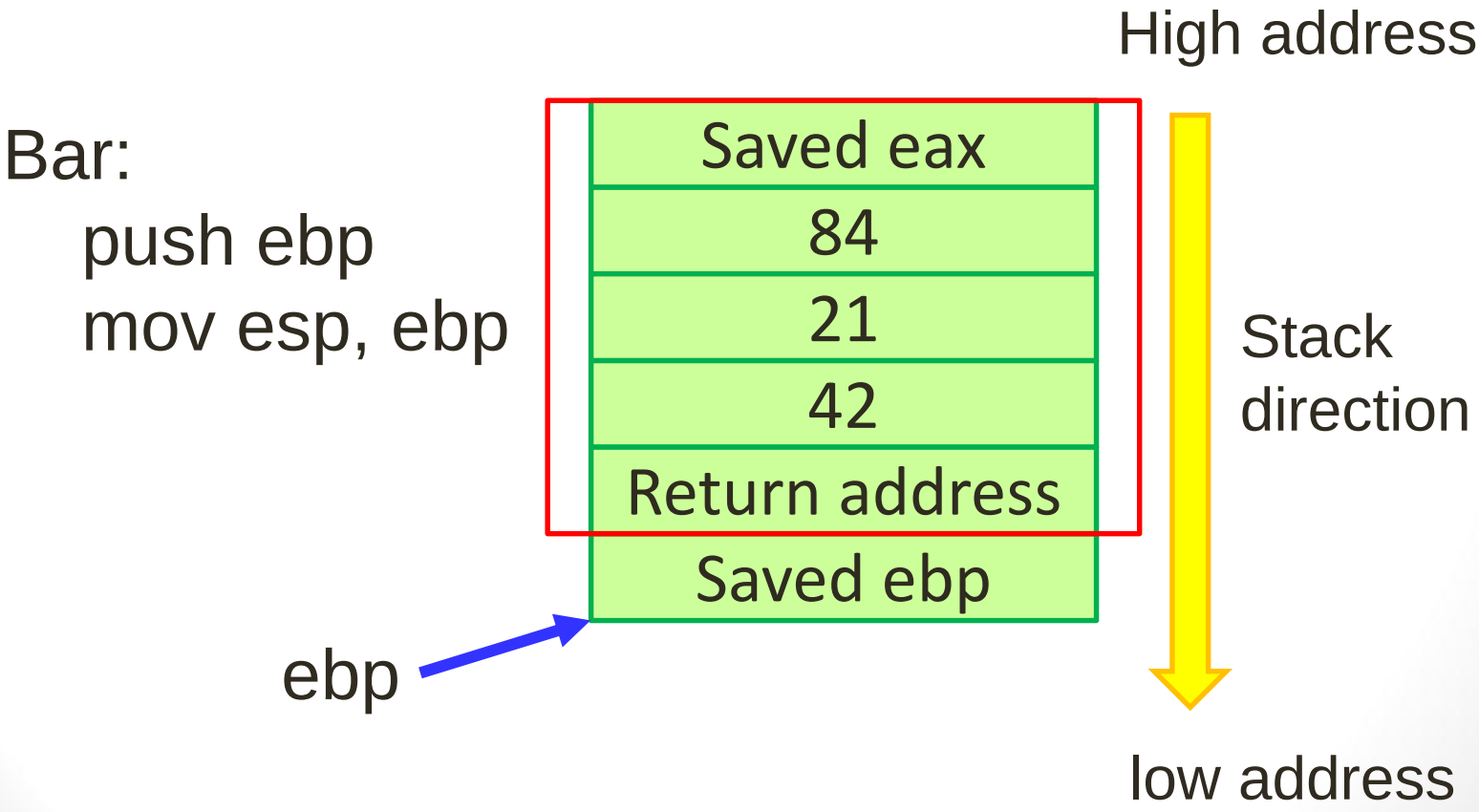
- The caller calls the routine. Doing the call will push the return address (current eip) on the stack.

call Bar



IA32 Calling Convention (5)

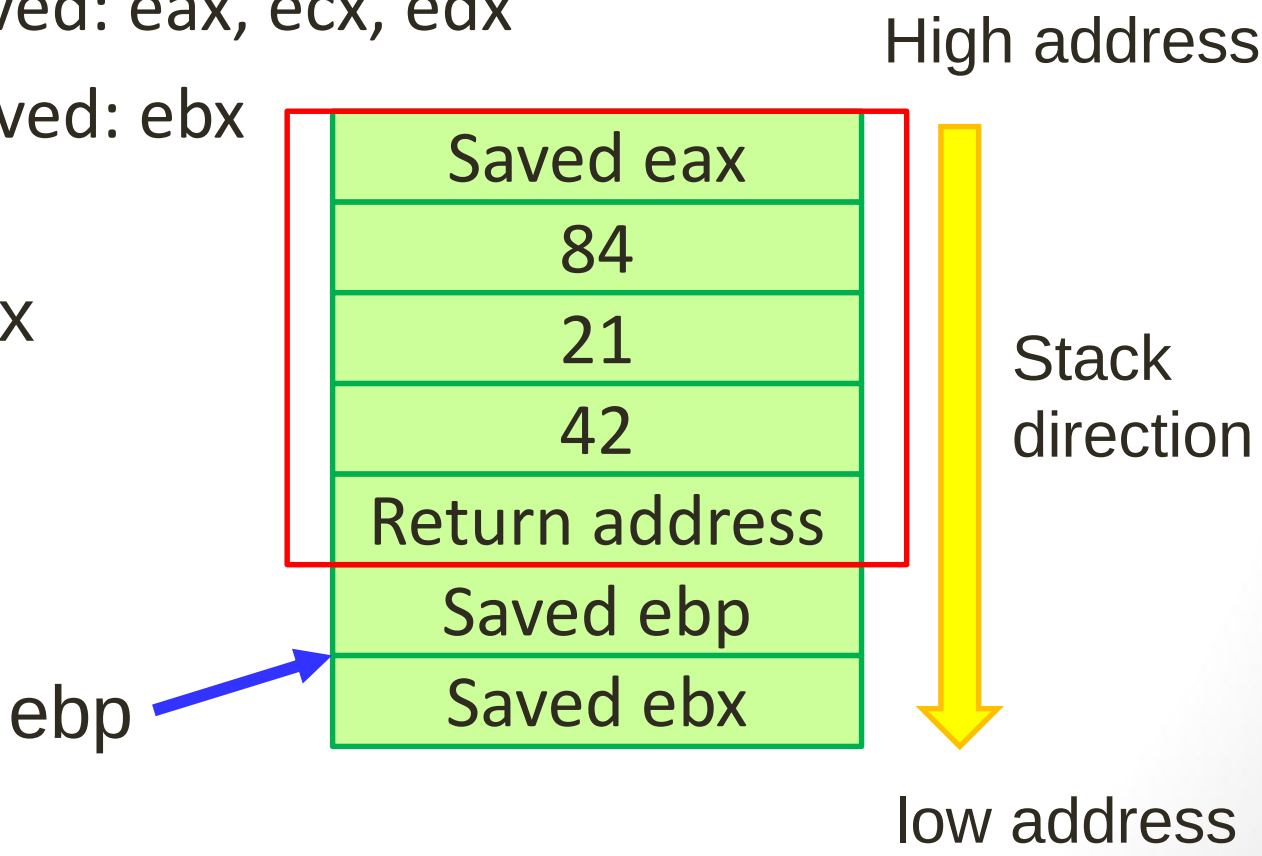
- The callee sets up a new stack frame. This is done by saving the ebp register and then setting it with the current content of the esp register.



IA32 Calling Convention (6)

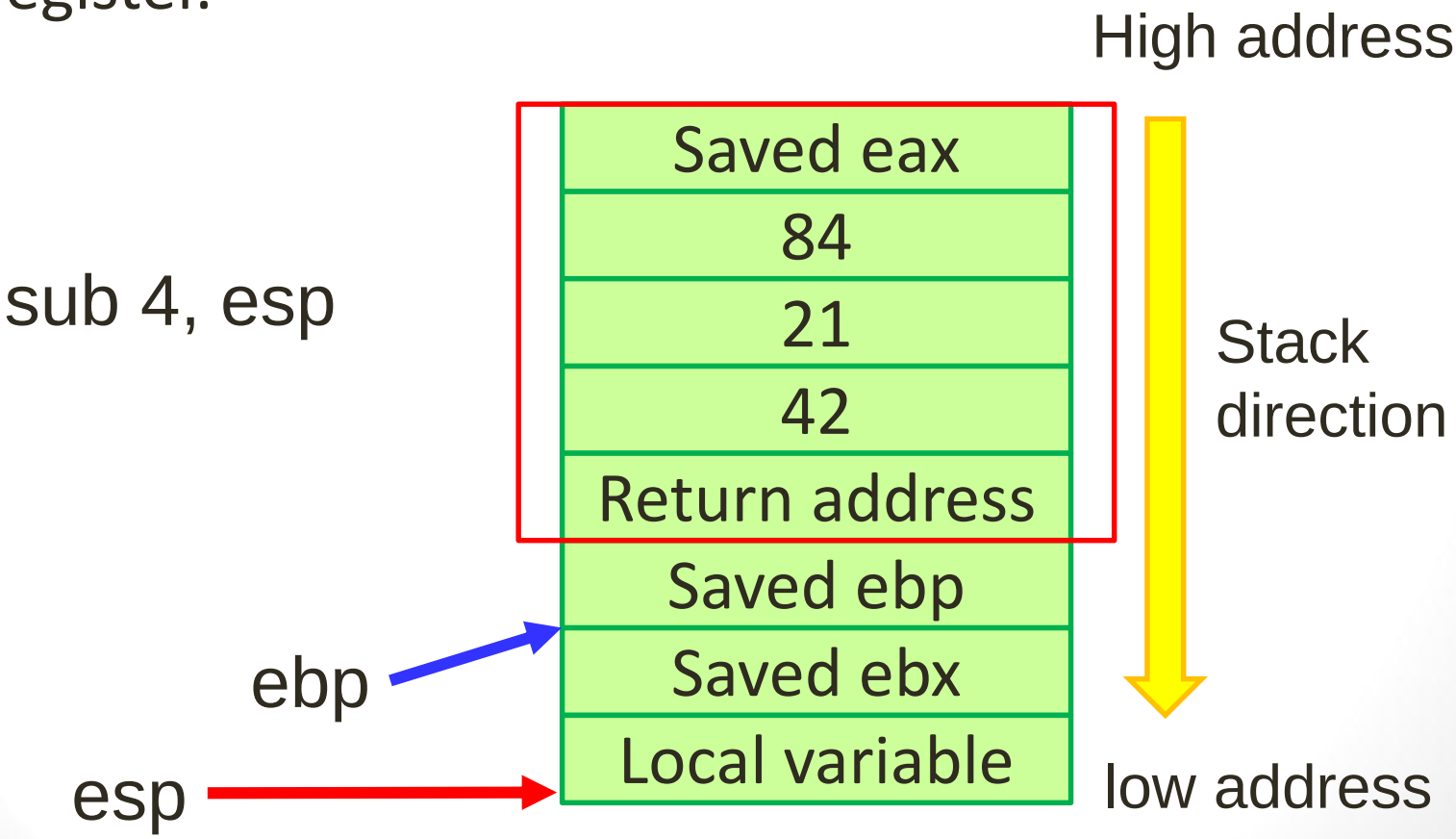
- The callee saves any register that will be used later by pushing their values on the stack.
- Caller saved: eax, ecx, edx
- Callee saved: ebx

push ebx



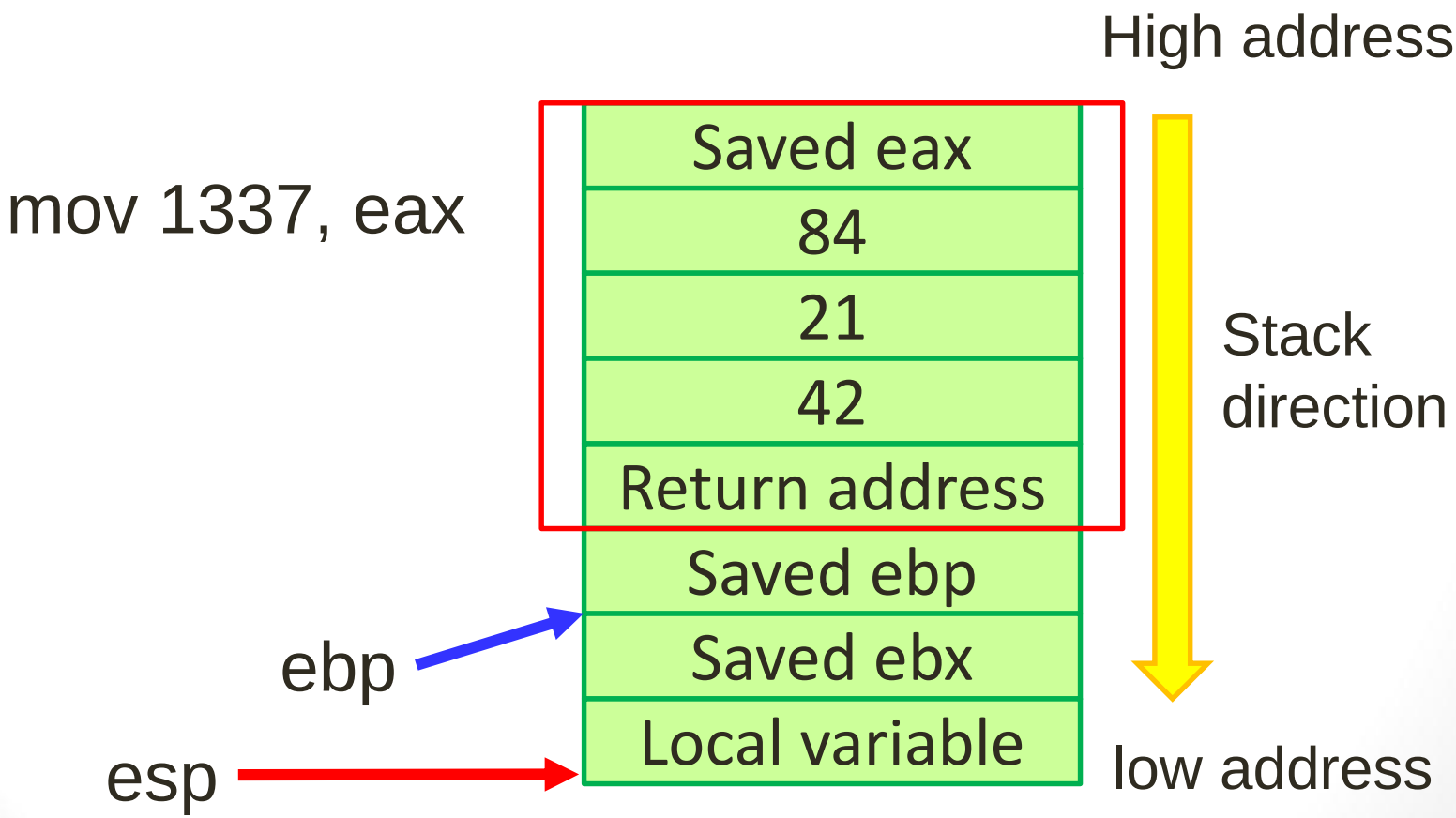
IA32 Calling Convention (7)

- The callee allocates room on the stack for local variables. This is done by decrementing the esp register.



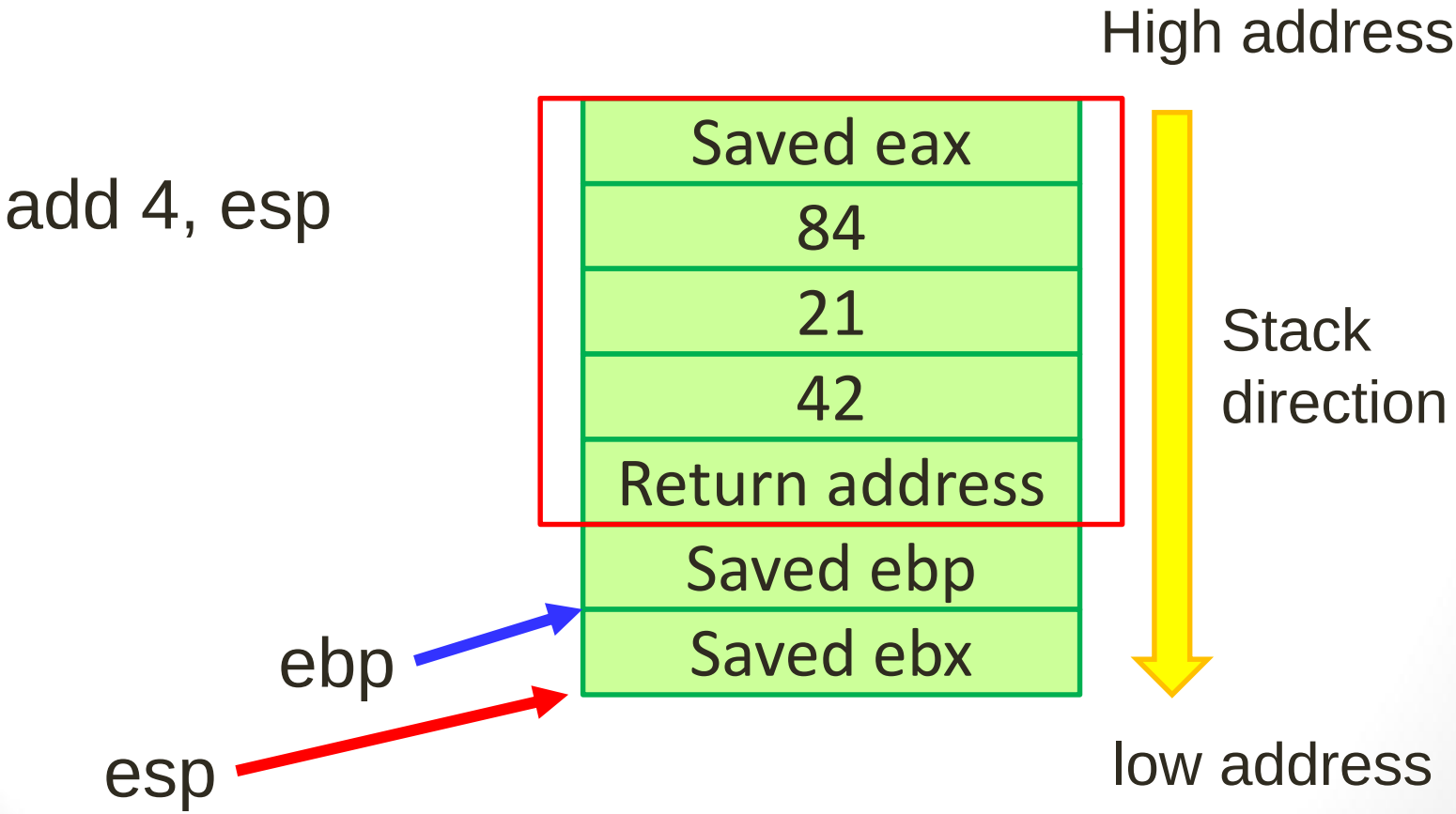
IA32 Calling Convention (8)

- The callee stores the return value in the eax register.



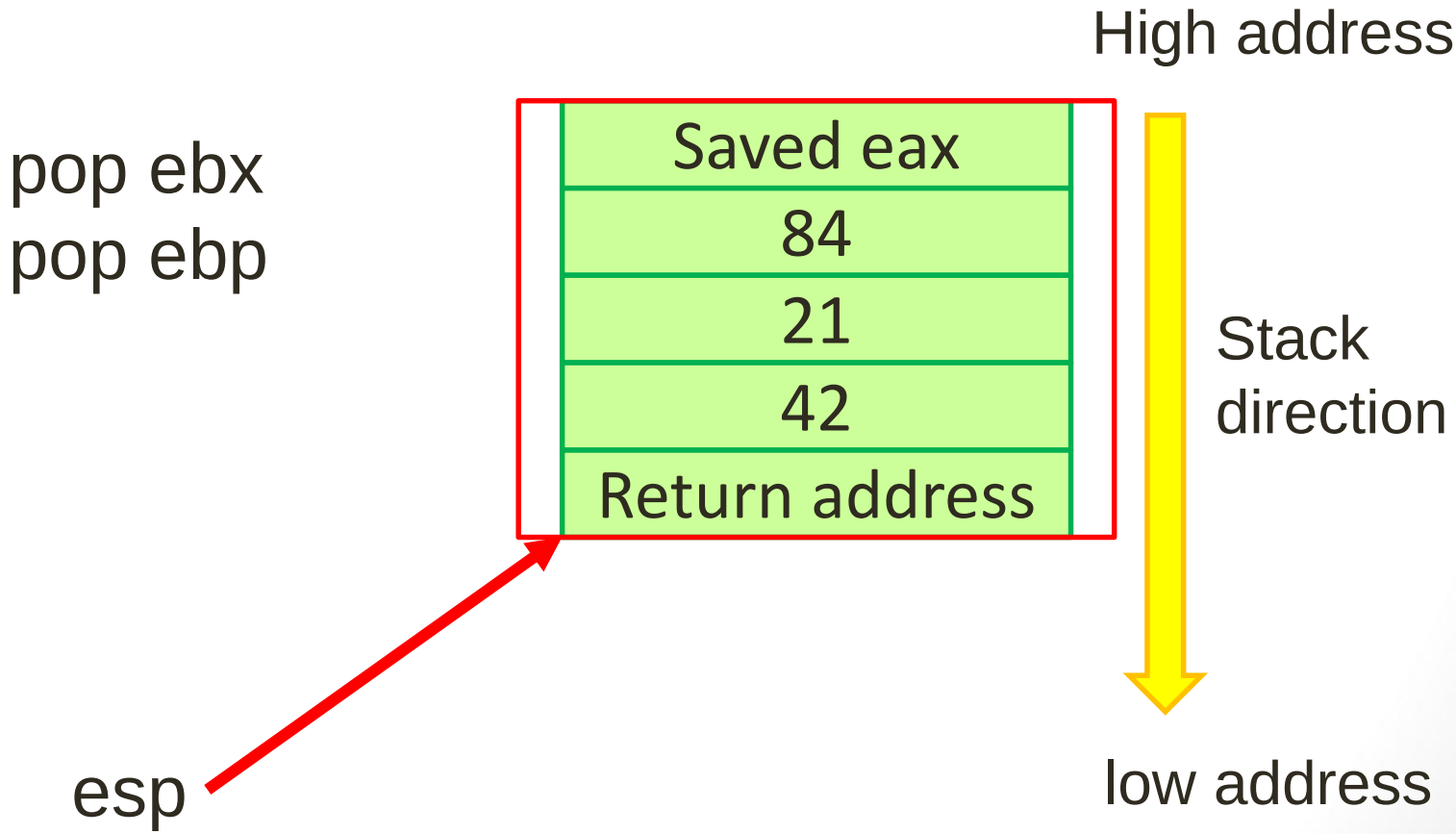
IA32 Calling Convention (9)

- The callee releases allocated space on the stack by incrementing the esp register.



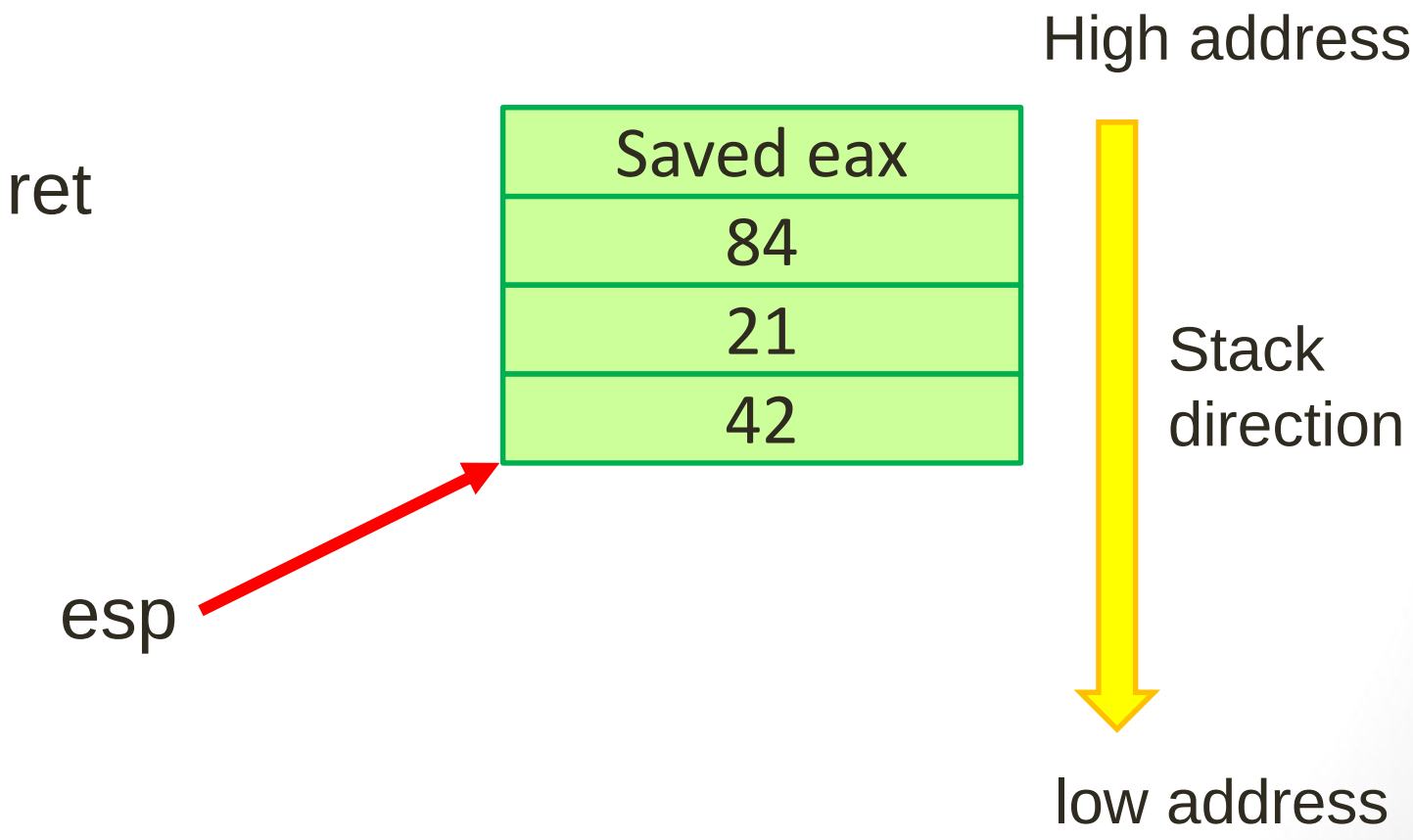
IA32 Calling Convention (10)

- The callee restores the registers content, including the ebp register.



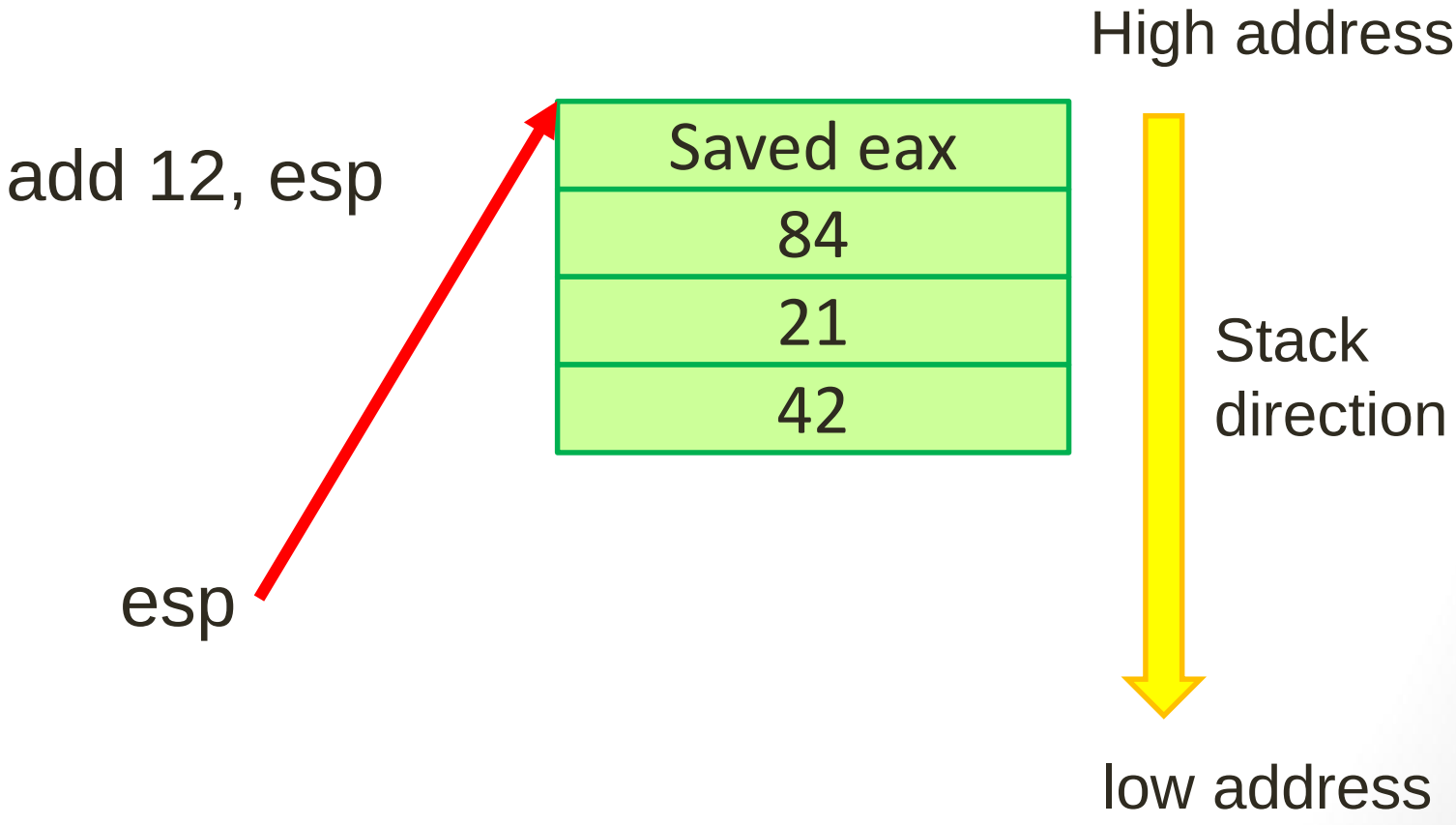
IA32 Calling Convention (11)

- The callee returns (this will pop the old value of eip).



IA32 Calling Convention (12)

- The caller must clean up the stack (i.e, remove the parameters by incrementing esp).



X86-64 Calling Convention (Linux) (1)

- From left to right, pass as many parameters as will fit in registers.
- The order in which registers are allocated, are:
  - For integers and pointers, rdi, rsi, rdx, rcx, r8, r9.
  - For floating-point (float, double), xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7
- Additional parameters are pushed on the stack, right to left, and are removed by the caller after the call.

X86-64 Calling Convention (Linux) (2)

- **The stack pointer %rsp MUST be aligned to a 16-byte boundary before making a call**.
- The callee-saved registers: **rbp, rbx, r12, r13, r14, r15**.
- The callee is also supposed to save the control bits of the XMCSR and the x87 control word, but x87 instructions are rare in 64-bit code so you probably don't have to worry about this.
- Integers are returned in **rax** or **rdx:rax**, and floating point values are returned in xmm0 or xmm1:xmm0.

X86 Assembly Code (1)

```
    .text
    .globl   main
    .type    main, @function
main:
    pushq %rbp
    movq  %rsp,%rbp



    popq   %rbp
```

使用64bit register

X86 Assembly Code (2)

- .comm symbol , length, alignment
  - .comm declares a common symbol named symbol.
  - ld will allocate *length* bytes of uninitialized memory
  - alignment is the desired alignment of the symbol, specified as a byte boundary

X86 Assembly Code (3)

```
int a, b;

int main(void)
{
    return 0;
}
```

X86 Assembly Code (32-bits)

```
.common a,4,4
.common b,4,4

  .text
      .globl   main
      .type    main, @function
main:
     pushl %ebp
     movl  %esp,%ebp
     movl  $0,%eax
     popl  %ebp
     ret
```

X86 Assembly Code (64bits)

```
.common a,4,4
.common b,4,4

  .text
      .globl   main
      .type    main, @function
main:
     pushq %rbp
     movq  %rsp,%rbp
     movl  $0,%eax
     popq  %rbp
     ret
```

X86 Assembly Code (4)

```
int a;
char b;

int main(void)
{
    a = 10;
    b = 3;
}
```

X86 Assembly Code (32-bits)

```
.common a,4,4
.common b,1,1

  .text
      .globl   main
      .type    main, @function
main:
     pushl %ebp
     movl  %esp,%ebp
     movl  $10,a
     movb  $3,b
     popl  %ebp
     ret
```

X86 Assembly Code (64bits)

```
.common a,4,4
.common b,1,1

  .text
      .globl   main
      .type    main, @function
main:
     pushq %rbp
     movq  %rsp,%rbp
     movl  $10,a(%rip)
     movb  $3,b(%rip)
     popq  %rbp
     ret
```

Grammar (1)

```
statement
    : Identifier '=' arith_expression
    |  IF '(' arith_expression ')‘
       if_then_statements
    ;

arith_expression
    : multExpr
      ( ‘+’ multExpr
      | ‘-’ multExpr
      )*
    ;
```

Grammar (2)

```
multExpr
      : signExpr
        ( '*' signExpr
        | '/' signExpr
        )*
      ;

signExpr
      : primaryExpr
      | '-' primaryExpr
      ;
```

Grammar (3)

```
primaryExpr
    : Integer_constant
    | Floating_point_constant
    | Identifier
    | '(' arith_expression ')'
    ;
```

C subset to x86 assembly

- Deliver information
  - Synthesized attributes
  - Inherited attributes

- Register number還需要適當的對應到x86 registers
  Ex: 1 => eax 5 => r8d
  2 => ebx 6 => r9d **(自己規劃與設計)**
  3 => ecx …
  4 => edx

```
@members {
    boolean TRACEON = false;
    HashMap<String,Integer> symtab = new
HashMap<String,Integer>();


    List<String> DataCode = new ArrayList<String>();
    List<String> TextCode = new ArrayList<String>();

    public static register reg = new register(0, 10);
    ...
}
```

```
primaryExpr returns [int attr_type, int reg_num]
    : Integer_constant
      {
         attr_type = 1;

         /* code generation */
         reg_num = reg.get();  /* get an register */
         TextCode.add("\t movl " + ...);
      }
    | Floating_point_constant { $attr_type = 2; }
    | Identifier
      {
         attr_type = symtab.get($Identifier.text);

         /* code generation */
         reg_num = reg.get(); /* get an register */
         TextCode.add("\t movl " + ...);
      }
    | '(' arith_expression ')'
    ;
```

```
multExpr returns [int attr_type, int reg_num]
      : a = signExpr { attr_type = $a.attr_type;
reg_num = $a.reg_num; }
       ( '*' signExpr
       | '/' signExpr
         )*
      ;


signExpr returns [int attr_type, int reg_num]
      : primaryExpr { attr_type =
$primaryExpr.attr_type; reg_num =
$primaryExpr.reg_num; }
      | '-' primaryExpr
      ;
```

```
arith_expression returns [int attr_type, int reg_num]
      : a = multExpr { attr_type = $a.attr_type;
reg_num = $a.reg_num; }
      ( '+' b = multExpr
        {
         /* code generation */
         TextCode.add("\t addl " + "\%" + $b.reg_num
+ ", \%" + reg_num);
        }
      | '-' c = multExpr
      )*
      ;
```

```
statement returns [int attr_type, int reg_num]
   : Identifier '=' arith_expression ';'
     {
       if (symtab.containsKey($Identifier.text)) {
          attr_type = symtab.get($Identifier.text);
       } else {
          /* Add codes to handle this error */

          attr_type = -2;
       }

       /* code generation */
       /* 根據attr_type，選擇適當的指令 */
       TextCode.add("\t movl " + ...);
     }
   ;
```

Support Function: printf()

- Function parameters:
  - For integers and pointers: rdi, rsi, rdx, rcx, r8, r9.

- printf("hello world\n");

  rdi

- printf("%d\n", var);

  rdi rsi

X86 Assembly Code (4)

```
int main(void)
{
   printf("Hello World\n");
}
```

X86 Assembly Code (64bits)

```
  .section .data
L1:
  .string "Hello World\n"
  .text
      .globl   main
      .type    main, @function
main:
     pushq %rbp
     movq  %rsp,%rbp
     movq  $L1,%rdi
     call  printf
     popq  %rbp
     ret
```

X86 Assembly Code (5)

```
int a;

int main(void)
{
   printf("%d\n", a);
}
```

X86 Assembly Code (64bits)

```
  .section .rodata
L1:
  .string "%d\n"
.common a,4,4
  .text
      .globl    main
      .typemain, @function
main:
    pushq %rbp
    movq  %rsp,%rbp
    movl  a(%rip),%esi    // arg2
    movq  $L1,%rdi        // arg1
    xor   %rax,%rax       // varargs
    call  printf
    popq  %rbp
    ret
```

EAX counts # of non-integer arguments being passed

GCC Options: -S

Use GCC to generate assembly code

- gcc –S –fno-asynchronous-unwind-tables test.c

- "-fno-asynchronous-unwind-tables" does not result in an extended EH (exception handling) section even when compiles applications written on "C".

Use ANTLR from the command-line (1)

- `$ java -cp antlr-3.4-complete.jar org.antlr.Tool myCompiler.g`

- 產生
  - myCompilerLexer.java
  - myCompilerParser.java
  - myCompilertokens

Use ANTLR from the command-line (2)

- **Compile**
  - `$javac -cp ./antlr-3.4-complete.jar myCompilerLexer.java myCompilerParser.java myCompiler_test.java`
- **Execute your compiler**
  - `$java -cp ./antlr-3.4-complete.jar:. myCompiler_test input.c`

  (產生input.s)

Use ANTLR from the command-line (3)

- **Execute your assembly code**
  - `$gcc input.s` (產生a.out)
  - `$./a.out`

Final Project

- **To define the subset of the language which you want to choose from C.**
- Give a set of testing programs which can illustrate the features of your testing programs. (at least 3 test programs and the generated assembly codes)
- Use the "**ANTLR**" to help you develop your compiler.
- You can use **C** or **Java** to write your compiler. (Java is recommended)
- Please ensure your program can be executed under the **mcore8** or **linux.cs.ccu.edu.tw** workstations.
- Support (at least two parameters) *printf* function in your compiler.
  - Ex: printf("%d\n", var);

Backup