Introduction to Compiler Design

Lesson 15:

Runtime Access to Variables

Roadmap

- Last

  Parameter-passing conventions

- Now
  - How do we deal with variables and scope?
  - How do we organize activation records?
  - How do we retrieve values of variables from activation records?

Scope

- We mostly worry about 3 flavors
  - Local
    - Declared and used in the same function
    - Further divided into “block” scope in Moo
  - Global
    - Declared at the outermost level of the program
  - Non-local (i.e., from nested scopes)
    - For static scope: variables declared in an outer scope
    - For dynamic scope: variables declared in the calling context

Local Variables: Examples

- What are the local variables here?

```
int fun(int a, int b){
    int c;
    c  = 1;
    if (a == 0){
        int d;
        d = 4;
    }
}
```

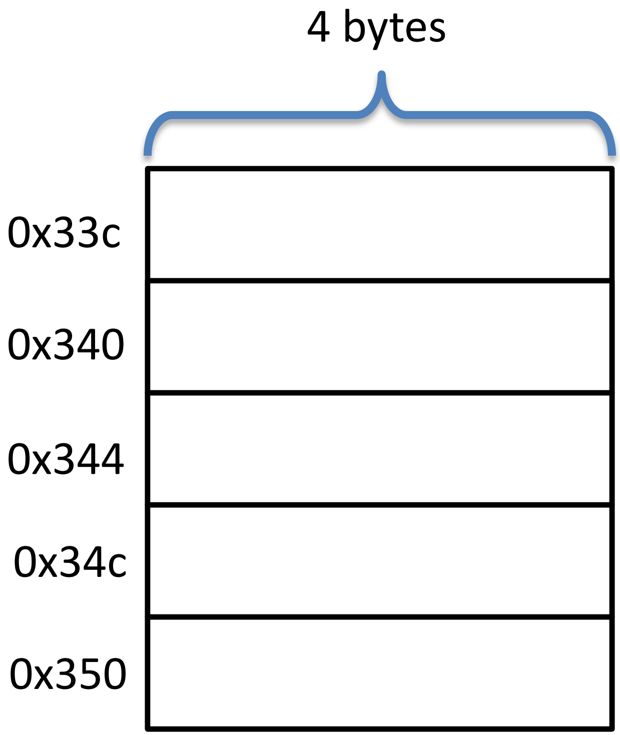
How Do We Access the Stack?

- Need a little MIPS knowledge
  - MIPS tutorial comes next
  - General anatomy of a MIPS instruction

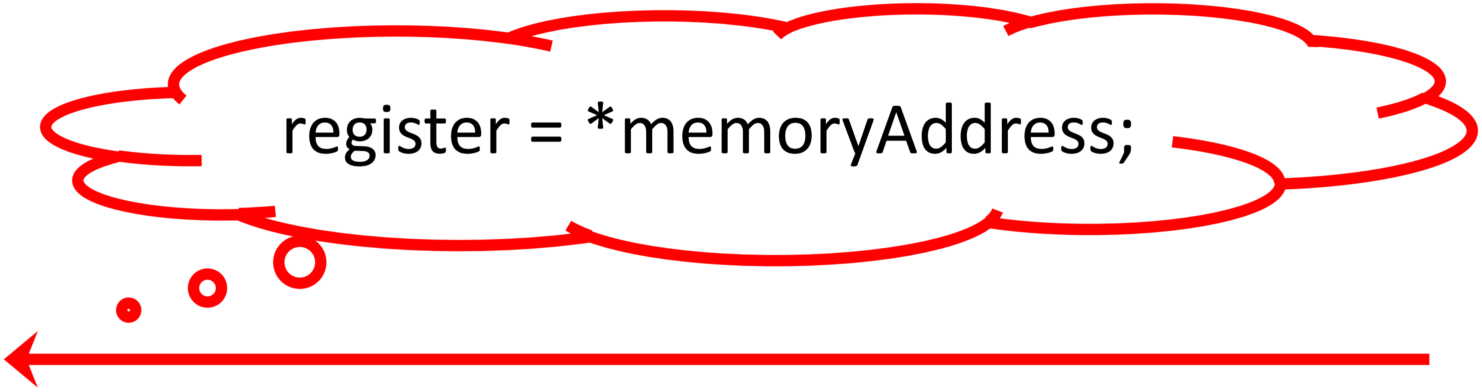
```
opcode Operand1 Operand2
```

How Do We Access the Stack?

- Use “load” and “store”instructions
  - Recall that every memory cell has an address
  - Calculate that memory address, then move data from/to that address

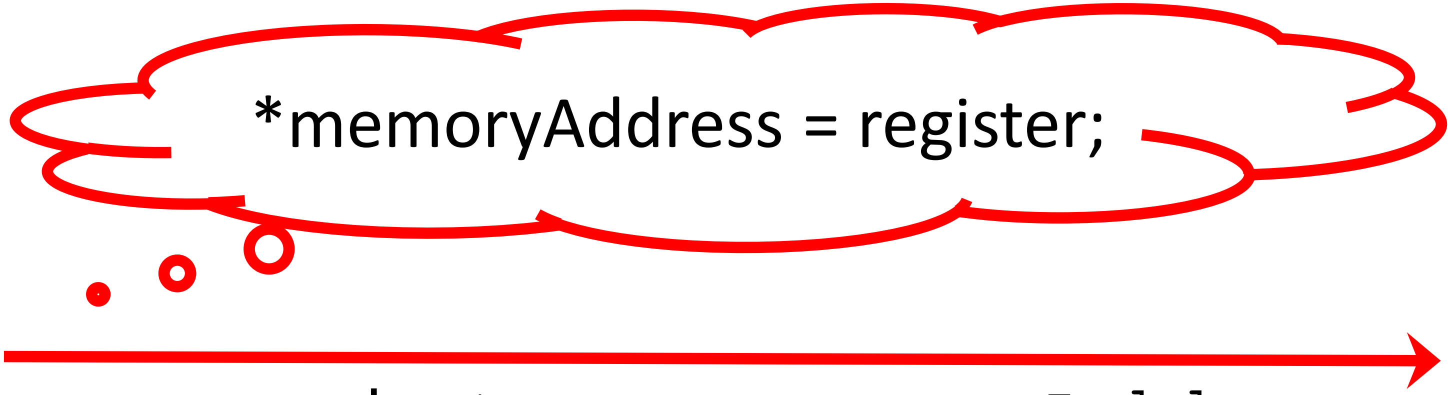


Basic Memory Operations



register = *memoryAddress;

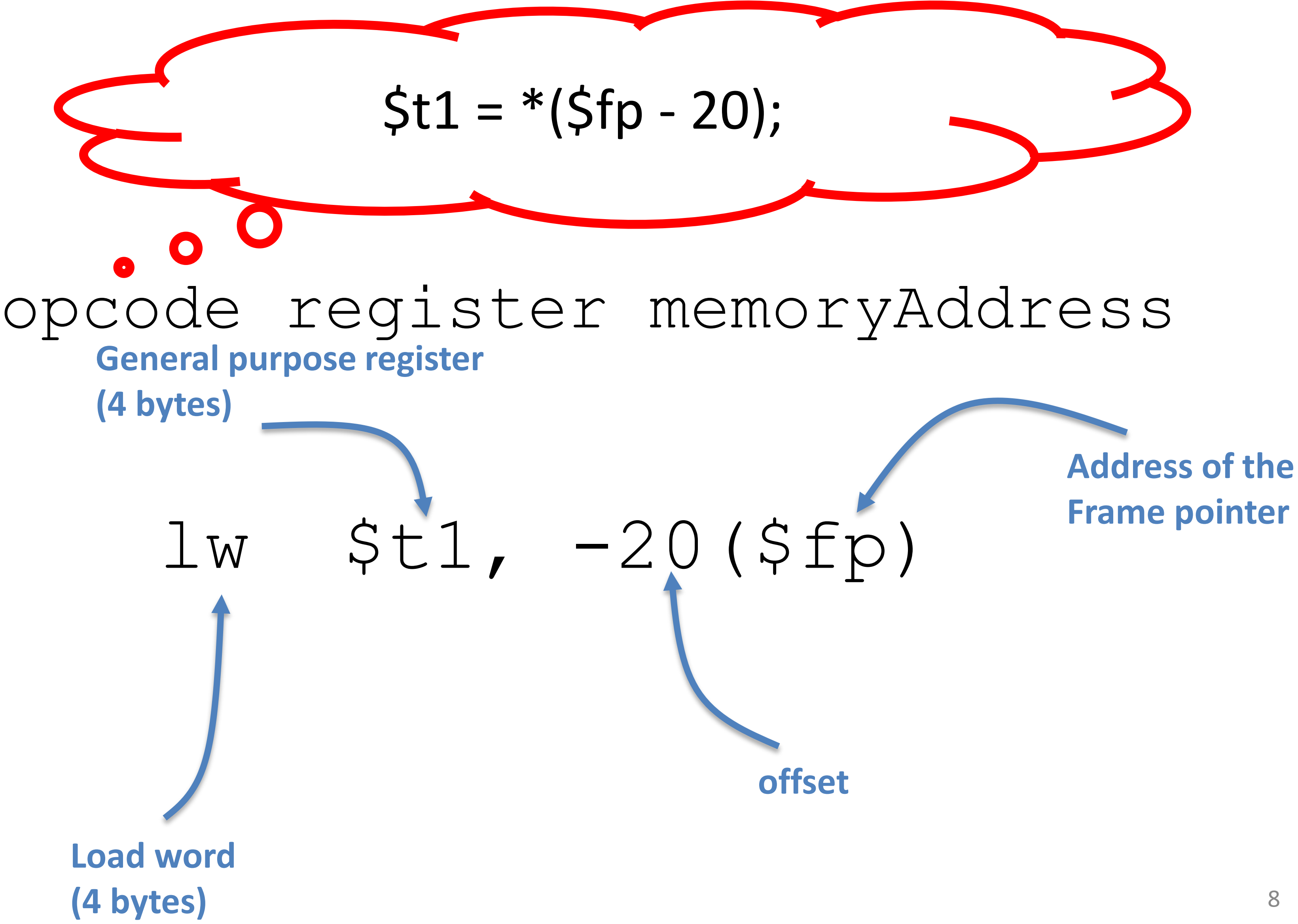
```
lw register memoryAddress
```



*memoryAddress = register;

```
sw register memoryAddress
```

Load-Word Example



$t1 = *($fp - 20);

```
opcode register memoryAddress
```

General purpose register
(4 bytes)

Address of the
Frame pointer

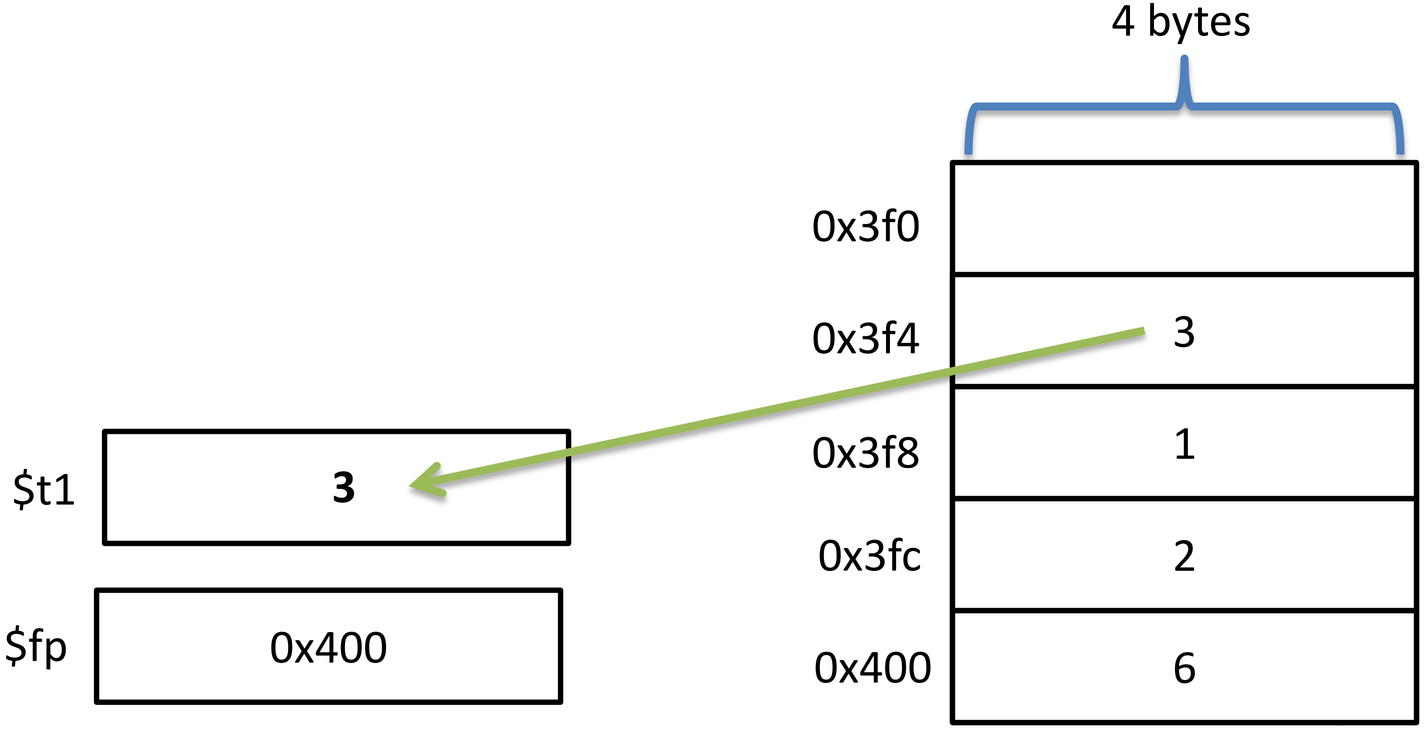
```
lw  $t1, -20($fp)
```

offset

Load word
(4 bytes)

Load Word in Action

```
lw  $t1, -12($fp)
```



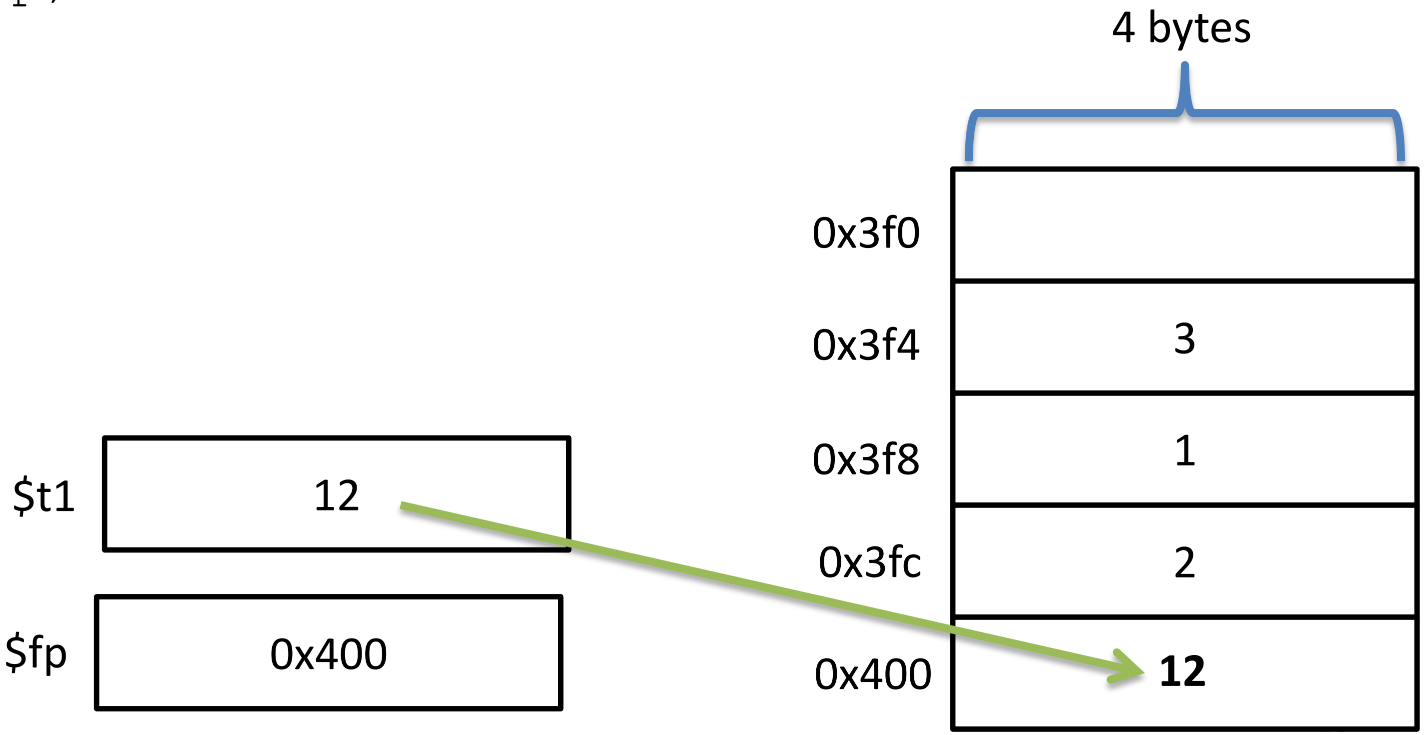
4 bytes

0x3f0	
0x3f4	3
0x3f8	1
0x3fc	2
0x400	6

$t1	**3**
$fp	0x400

Store Word in Action

```
sw  $t1, 0($fp)
```



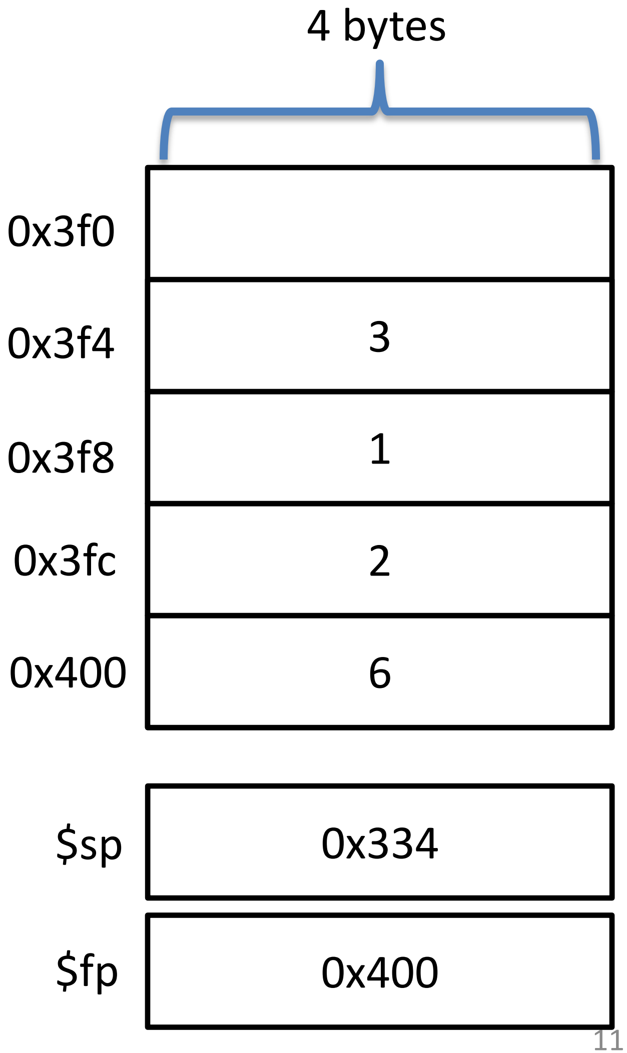
4 bytes

0x3f0	
0x3f4	3
0x3f8	1
0x3fc	2
0x400	**12**

$t1	12
$fp	0x400

Relative Access for Locals

- Why do we access locals from $fp?
  - That’s where the activation record starts
- What if we used $sp instead?
  - We don’t know how many locals



4 bytes

0x3f0	
0x3f4	3
0x3f8	1
0x3fc	2
0x400	6

$sp	0x334
$fp	0x400

A Simple Memory-Allocation Scheme

- Reserve a slot for each variable in the function

```
int test (int x, int y){
  int a, b;
  if (x){
    int s;
  } else {
    int t, u, v;
    u = b + y;
  }
}
```

$sp	0x3d4
$fp	0x3f8

0x3d4	
0x3dc	(v)
0x3e0	(u)
0x3e4	(t)
0x3e8	(s)
0x3ec	(b)
0x3f0	(a)
0x3f4	(control link)
0x3f8	(return addr)
0x3fc	(x)
0x400	(y)

Simple Memory-Allocation Algorithm

```
For each function
Set offset = +4
for each parameter
    add name to symbol table
    offset += size of parameter
offset = -4

for each local
   offset -= size of variable
   add name to symbol table
```

Simple Memory-Allocation Implementation

- Add an offset field to each symbol table entry
- During name analysis, add the offset along with the name
- Walk the AST performing decrements at each declaration node

Handling Global Variables

- In a sense, globals easier to handle than locals
  - Space allocated directly at compile time instead of indirectly via **$fp** and **$sp** registers
  - Never needs to be deallocated
- Place in static data area
  - In MIPS, handling with a special storage directive
  - Variables referred to by name, not by address

Memory-Region Example

```
.data
_x: .word 10
_y: .byte 1
_z: .asciiz “I am a string”
.text
lw $t0, _x  #Load from x into $t0
sw $t0, _x  #Store from $to into x
```

Accessing Non-Local Variables

- Static scope
  - Variable declared in one procedure and accessed in a nested one
- Dynamic scope
  - Any variable x used that is not declared locally resolves to instance of x in the AR closest to the current AR

Example: Static Non-Local Scope

- Each function has its own AR
  - Inner function accesses the outer AR

```
function main(){
    int a = 0;

    function subprog(){
        a = a + 1;
    }
}
```

Memory Access: Static Non-Local Scope

```
void procA(){  // level 1
 int x, y;
 void procB(){ // level 2

  void procC(){ //level 3
   int z;
   void procD(){//level 4
    int x;
    x = z + y;
    procB();
   }
   x = 4;
   z = 2;
   procB();
   procD();
  }
  x = 3;
  y = 5;
}
```

Roadmap

- We learned about variable access
  - Local vs. global variables
  - Static vs. dynamic scopes

- Next
  - We’ll start getting into the details of MIPS
  - Code generation