

Assembler 3

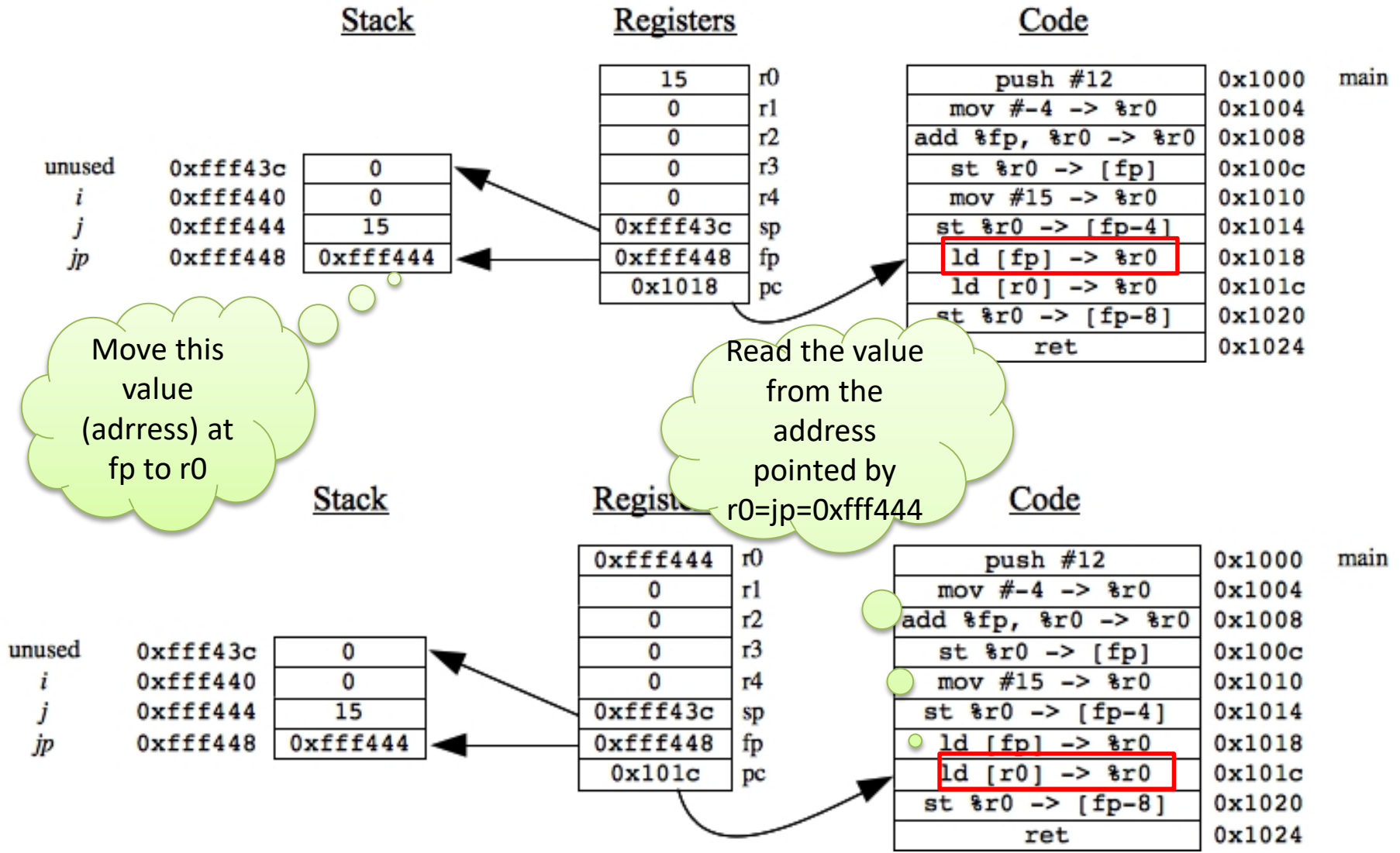
Pointers

C code	Assembly code
<pre>main() { int i, j, *jp; jp = &j; j = 15; i = *jp; }</pre>	<pre>main: push #12 mov #-4 -> %r0 / jp = &j. add %fp, %r0 -> %r0 st %r0 -> [fp] mov #15 -> %r0 / j = 15 st %r0 -> [fp-4] ld [fp] -> %r0 / i = *jp ld [%r0] -> %r0 st %r0 -> [fp-8] ret</pre>

Pointers

Execution of program

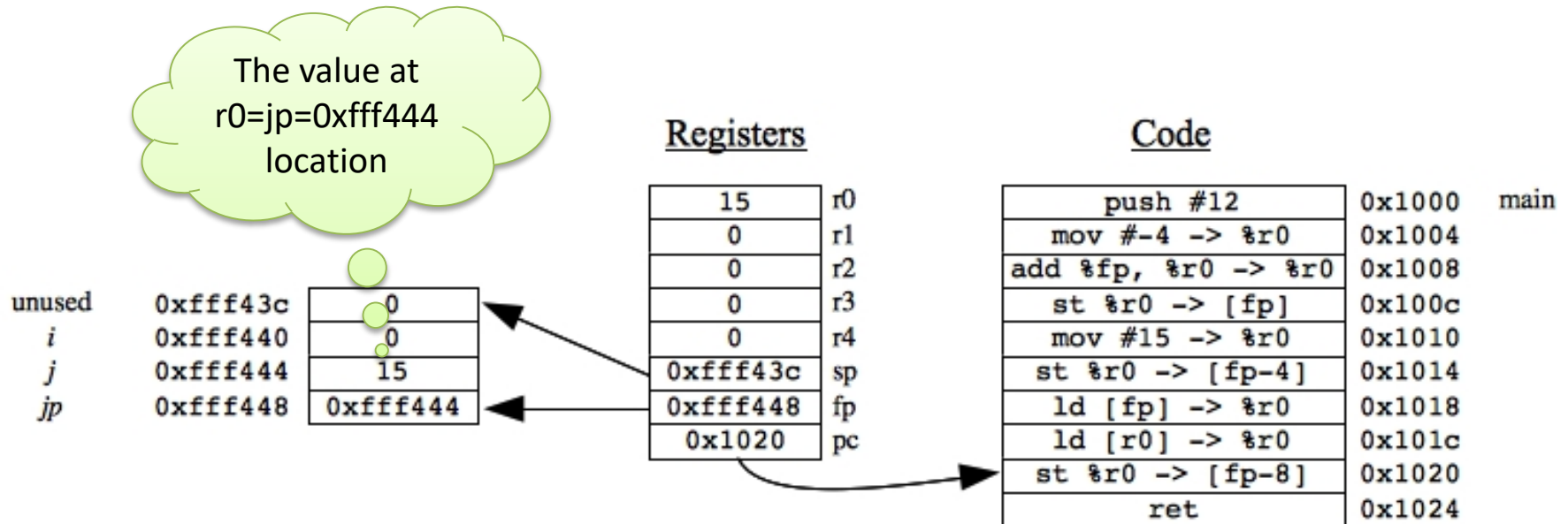
for $i = *jp$



Pointers

Execution of program

for **i** = *jp



Procedure with pointers

C kodu

```
int a(int *p)
{
    return *p;
}
```

```
main()
{
    int i, j;

    j = 15;
    i = a(&j);
}
```

Assembly kod

```
a:
    ld [fp+12] -> %r0    / get p's value
    ld [r0] -> %r0      / dereference it
    ret

main:
    push #8

    mov #15 -> %r0      / j = 15
    st %r0 -> [fp]

    st %fp -> [sp]--    / push &j on the stack
    jsr a               / and call a()
    pop #4
    st %r0 -> [fp-4]

    ret
```

Array operations

Array operations

C code

```
int main()
{
    int array[5];

    array[0] = 10;
    array[1] = 11;
    array[2] = 12;
    array[3] = 2;
    array[4] = 15;

    a(array);
}
```

Assembly code

```
main:
    push #20

    mov #10 -> %r0          / Store the values of array
    st %r0 -> [fp-16]
    mov #11 -> %r0
    st %r0 -> [fp-12]
    mov #12 -> %r0
    st %r0 -> [fp-8]
    mov #2 -> %r0
    st %r0 -> [fp-4]
    mov #15 -> %r0
    st %r0 -> [fp]

    mov #-16 -> %r0         / Push array onto the stack
    add %fp, %r0 -> %r0
    st %r0 -> [sp]--
    jsr a                   / call a
    pop #4
    ret
```

Array operations

```
main()
{
    int array[5];
    array[0] = 10;
    array[1] = 11;
    array[2] = 12;
    array[3] = 2;
    array[4] = 15;

    a(array);
}
```

main:

push #20

mov #10 -> %r0

st %r0 -> [fp-16]

mov #11 -> %r0

st %r0 -> [fp-12]

mov #12 -> %r0

st %r0 -> [fp-8]

mov #2 -> %r0

st %r0 -> [fp-4]

mov #15 -> %r0

st %r0 -> [fp]

mov #-16 -> %r0

add %fp, %r0 -> %r0

st %r0 -> [sp]--

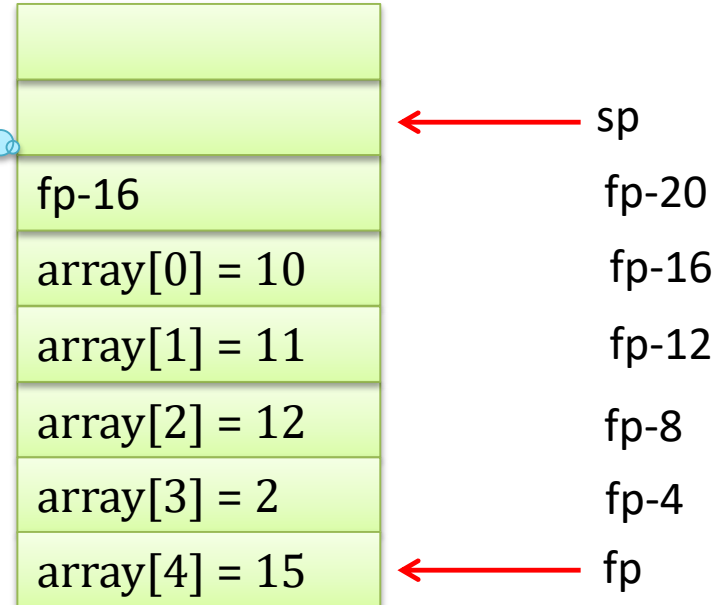
jsr a / call a

pop #4

ret

Stack before
executing
jsr a

Stack



Dizinin başlangıç adresi fp-16
adres değerini, array[0]
elemanının bir üstüne yaz.
Bunun için fp ile -16 toplanıyor ve
sonuç stack pointer ile gösterilen
fp-20 adresine kaydedildikten
sonra stack pointer azaltılarak
sıradaki boş yeri gösteriyor.

Array operations

JASS - CS360 Assembler

Stack

Step 14

CSR

Registers

Globals

Code

The address of the first element of the array is written to stack as a local parameter to function a.

Stack content before jsr a is executed.

fp-20
fp-16
fp-12
fp-8
fp-4
fp

0xffff430 0x0
0xffff434 0xffff438
0xffff438 0xa
0xffff43c 0xb
0xffff440 0xc
0xffff444 0x2
0xffff448 0xf

0x104c st %r0 -> [fp-16]
0x1050 mov #11 -> %r0
0x1054 st %r0 -> [fp-12]
0x1058 mov #12 -> %r0
0x105c st %r0 -> [fp-8]
0x1060 mov #2 -> %r0
0x1064 st %r0 -> [fp-4]
0x1068 mov #15 -> %r0
0x106c st %r0 -> [fp]
0x1070 mov #-16 -> %r0
0x1074 add %fp, %r0 -> %r0
0x1078 st %r0 -> [sp]--
0x107e jsr a
0x1080 pop #4
0x1084 ret

0xffff430 0x0
0xffff434 0xffff438
0xffff438 0xa
0xffff43c 0xb
0xffff440 0xc
0xffff444 0x2
0xffff448 0xf

0x104c st %r0 -> [fp-16]
0x1050 mov #11 -> %r0
0x1054 st %r0 -> [fp-12]
0x1058 mov #12 -> %r0
0x105c st %r0 -> [fp-8]
0x1060 mov #2 -> %r0
0x1064 st %r0 -> [fp-4]
0x1068 mov #15 -> %r0
0x106c st %r0 -> [fp]
0x1070 mov #-16 -> %r0
0x1074 add %fp, %r0 -> %r0
0x1078 st %r0 -> [sp]--
0x107e jsr a
0x1080 pop #4
0x1084 ret

Step

Animate

Make Animation Fast

Scroll Stack Up

Scroll Stack Down

Scroll Code Up

Scroll Code Down

Window Height

20

Redraw

Quit

Array operations

JASS -- CS360 Assembler

Stack

Step 15

CSR

Registers

r0	0xffff438
r1	0x0
r2	0x0
r3	0x0
r4	0x0
sp	0xffff428
fp	0xffff428
pc	0x1000

Code

0x1000	push #4	a
0x1004	ld [fp+12] -> %r0	
0x1008	ld [r0] -> %r0	
0x100c	st %r0 -> [fp]	
0x1010	ld [fp+12] -> %r0	
0x1014	mov #12 -> %r1	
0x1018	add %r0, %r1 -> %r0	
0x101c	ld [r0] -> %r0	
0x1020	st %r0 -> [fp]	
0x1024	ld [fp] -> %r0	
0x1028	mov #4 -> %r1	
0x102c	mul %r0, %r1 -> %r0	
0x1030	ld [fp+12] -> %r1	
0x1034	add %r0, %r1 -> %r0	
0x1038	ld [r0] -> %r0	
0x103c	st %r0 -> [fp]	
0x1040	ret	
0x1044	push #20	main
0x1048	mov #10 -> %r0	
0x104c	st %r0 -> [fp-16]	

0xffff428 0x0

0xffff42c 0xffff448

0xffff430 0x1080

0xffff434 0xffff438

0xffff438 0xa

0xffff43c 0xb

0xffff440 0xc

0xffff444 0x2

0xffff448 0xf

After jsr a is executed.

Step

Animate

Make Animation Fast

Scroll Stack Up

Scroll Stack Down

Scroll Code Up

Scroll Code Down

Window Height

20

Redraw

Quit

Array operations

C code

```
void a(int *p)
{
    int i;

    i = p[0];
    i = p[3];
    i = p[i];
}
```

Assembly code

```
a:
    push #4
    ld [fp+12] -> %r0      / i = p[0]
    ld [r0] -> %r0
    st %r0 -> [fp]

    ld [fp+12] -> %r0      / i = p[3]
    mov #12 -> %r1
    add %r0, %r1 -> %r0
    ld [r0] -> %r0
    st %r0 -> [fp]

    ld [fp] -> %r0         / i = p[i]
    mov #4 -> %r1
    mul %r0, %r1 -> %r0
    ld [fp+12] -> %r1
    add %r0, %r1 -> %r0
    ld [r0] -> %r0
    st %r0 -> [fp]

    ret
```

Array operations

```
a(int *p)
{
    int i;

    i = p[0];
    i = p[3];
    i = p[i];
}
```

a:

push #4

ld [fp+12] -> %r0

ld [r0] -> %r0

st %r0 -> [fp]

ld [fp+12] -> %r0

mov #12 -> %r1

add %r0, %r1 -> %r0

ld [r0] -> %r0

st %r0 -> [fp]

ld [fp] -> %r0

mov #4 -> %r1

mul %r0, %r1 -> %r0

ld [fp+12] -> %r1

add %r0, %r1 -> %r0

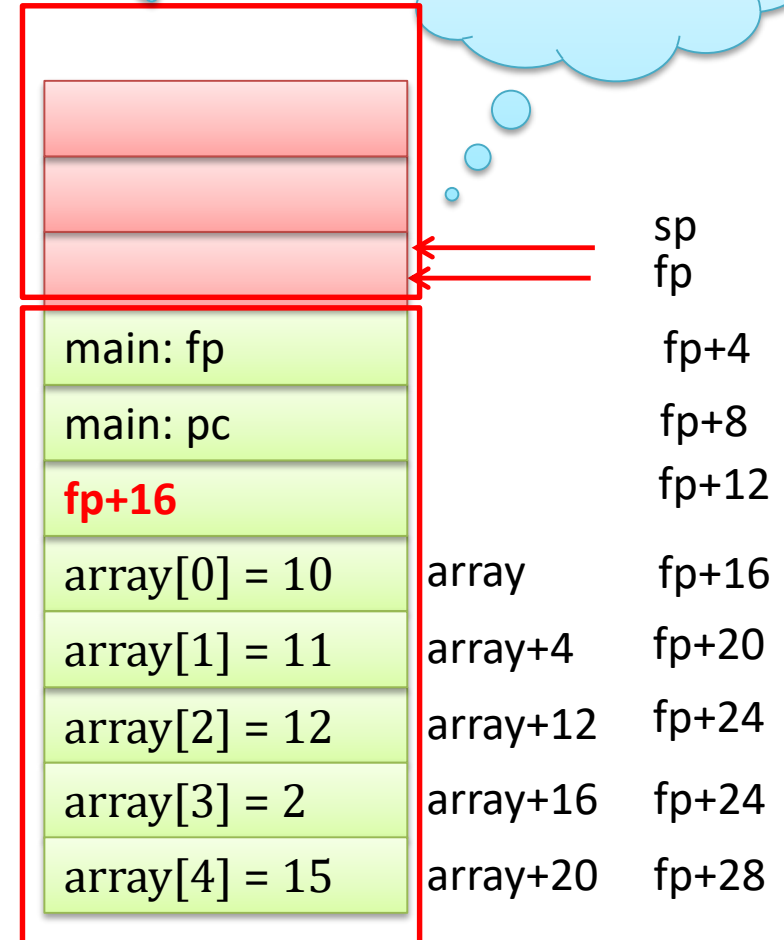
ld [r0] -> %r0

st %r0 -> [fp]

ret

fp shows new frame

Before **a** starts
executing



Array operations

a:

```
push #4
```

```
ld [fp+12] -> %r0
```

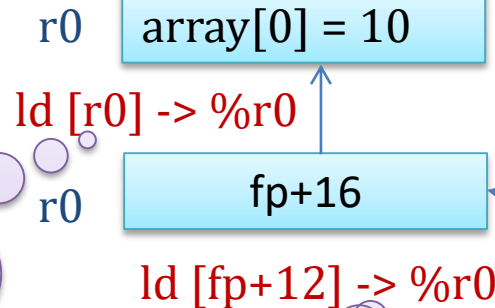
```
ld [r0] -> %r0
```

```
st %r0 -> [fp]
```

```
a(int *p)
{
    int i;

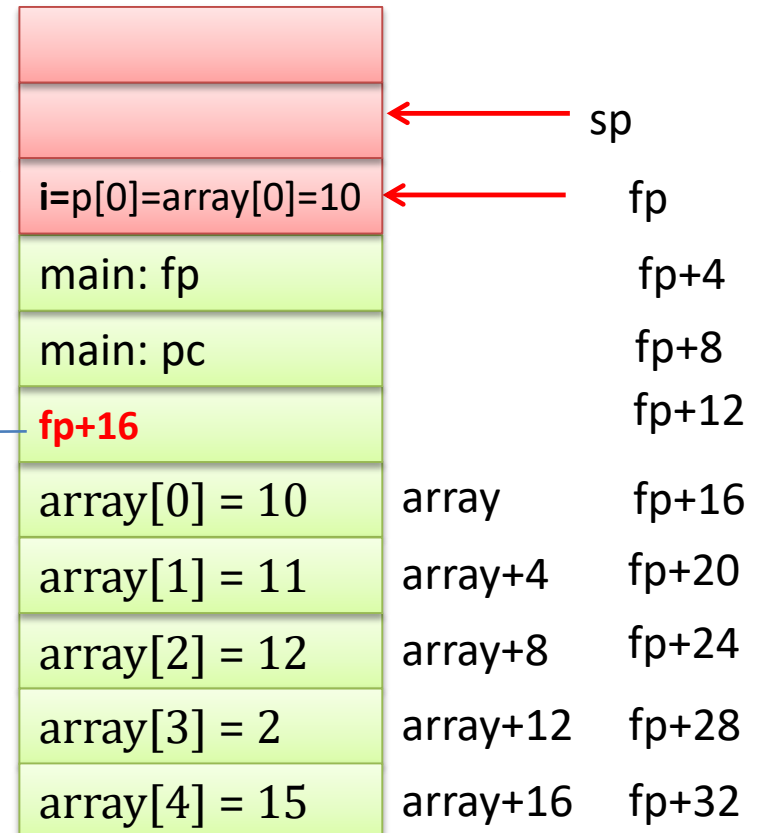
    i = p[0];
    i = p[3];
    i = p[i];
}
```

Stack after
executing;
int i;
i = p[0];



Load the data pointed by `r0=fp+16` to `r0` again. Since `fp+16` points to first element of the array, the value of `r0` is set to 10.

Load the data pointed by `fp+12` to `r0`. `fp+12` is the address of the local parameter in function `a`. Since local parameter is a pointer to array, it contains the address of first element of the array.



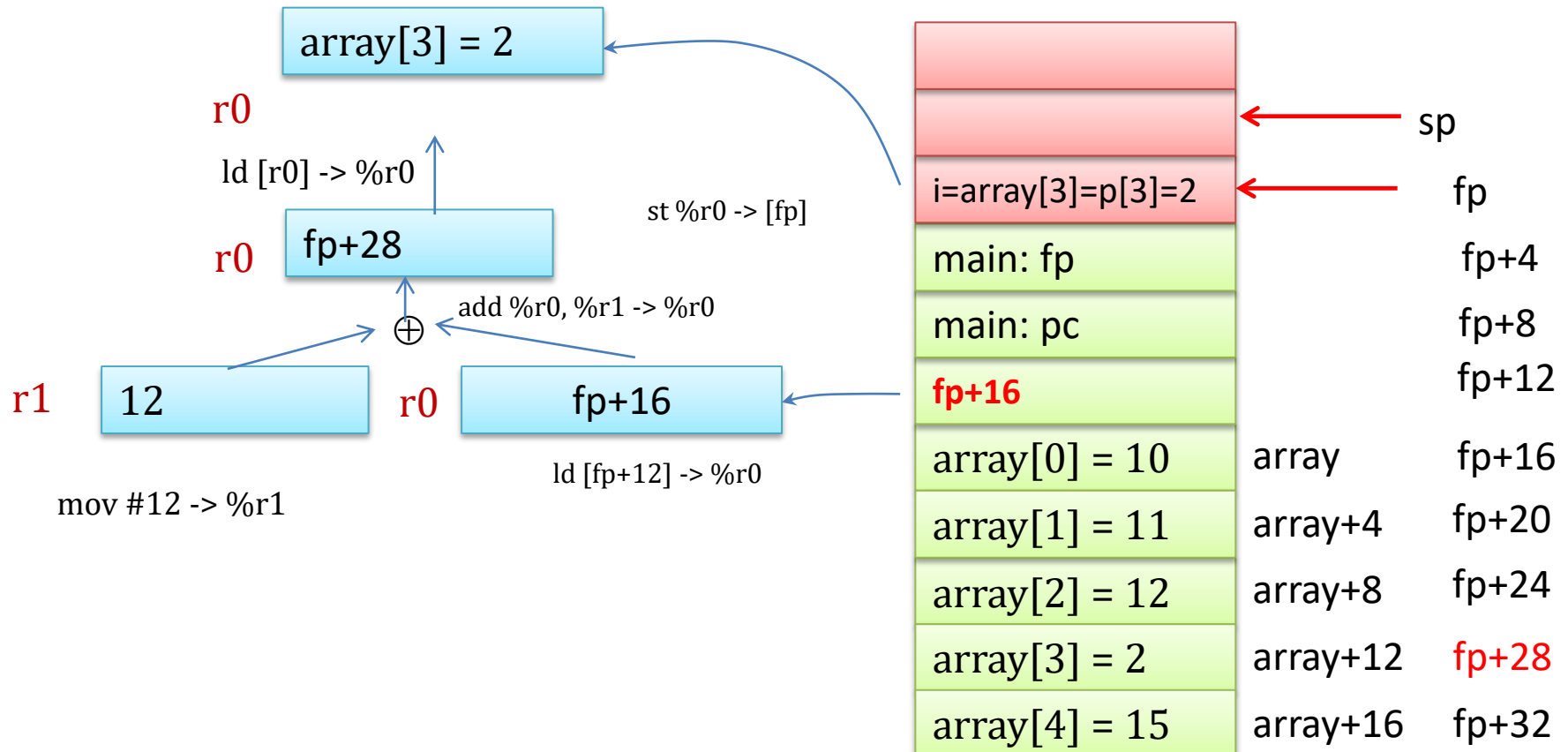
Array operations

```
ld [fp+12] -> %r0
mov #12 -> %r1
add %r0, %r1 -> %r0
ld [r0] -> %r0
st %r0 -> [fp]
```

```
a(int *p)
{
    int i;

    i = p[0];
    i = p[3];
    i = p[i];
}
```

Stack after
executing;
i = p[3];

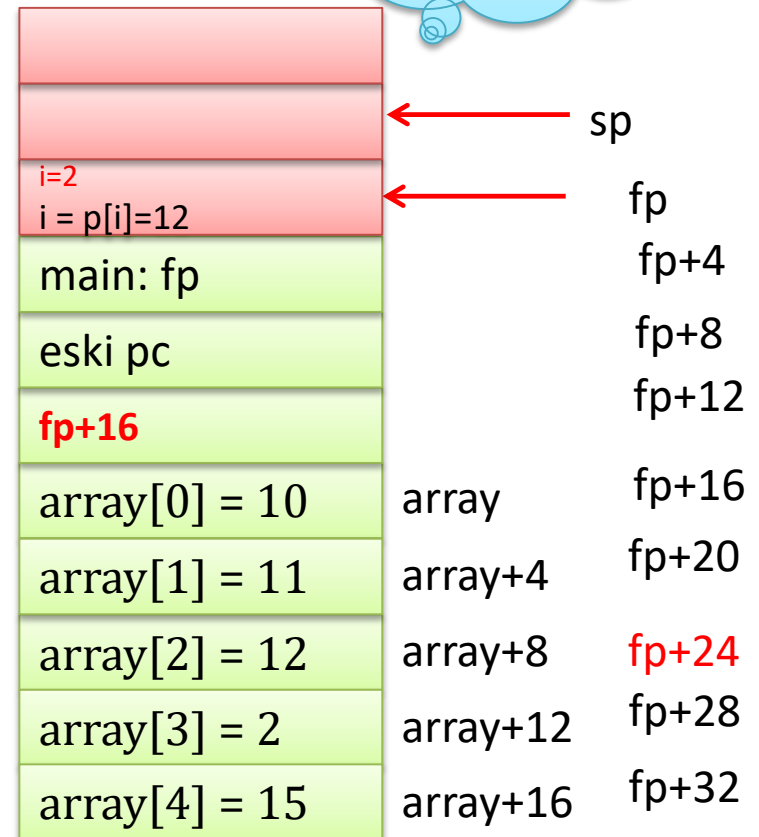
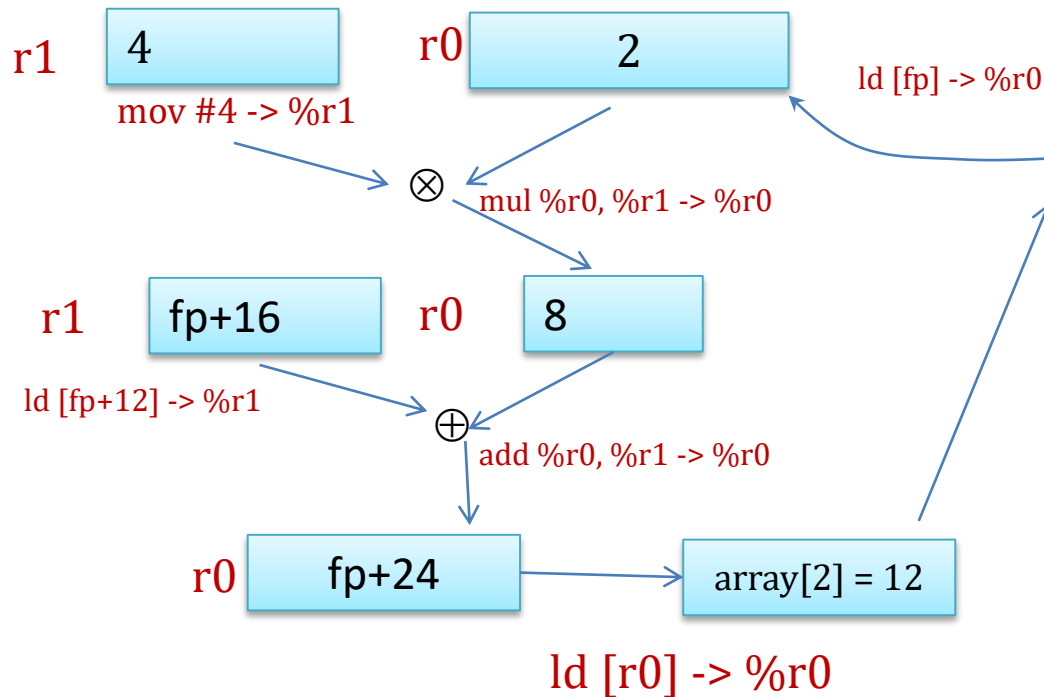


Array operations

```
ld [fp] -> %r0
mov #4 -> %r1
mul %r0, %r1 -> %r0
ld [fp+12] -> %r1
add %r0, %r1 -> %r0
ld [r0] -> %r0
st %r0 -> [fp]
```

```
a(int *p)
{
    int i;

    i = p[0];
    i = p[3];
    i = p[i];
}
```



Uninitialized Locals

C code

<pre>main() { int x; a(5, 6, 7, 8); x = b(2); }</pre>	<pre>a(int i,int j,int k,int l) { int m; m = i + j + k + l; }</pre>	<pre>int b(int i) { int p[5]; return p[i]; }</pre>
--	--	---

we don't initialize **p** and we simply return **p[i]**. It returns a value from the specified location.

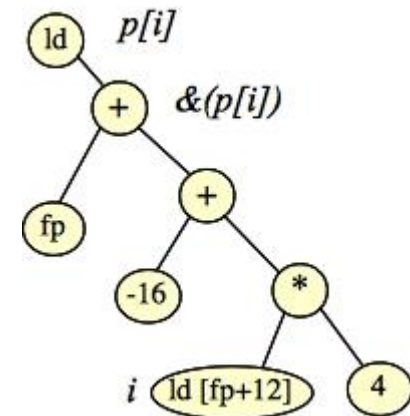
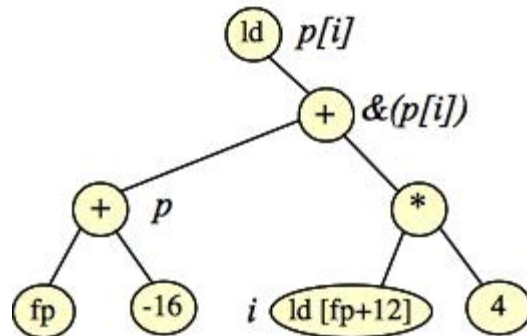
Assembly code

<pre>main: push #4 mov #8 -> %r0 st %r0 -> [sp]-- mov #7 -> %r0 st %r0 -> [sp]-- mov #6 -> %r0 st %r0 -> [sp]-- mov #5 -> %r0 st %r0 -> [sp]-- jsr a pop #16 mov #2 -> %r0 st %r0 -> [sp]-- jsr b pop #4 st %r0 -> [fp] ret</pre>	<pre>a: push #4 ld [fp+12] -> %r0 ld [fp+16] -> %r1 add %r0, %r1 -> %r0 ld [fp+20] -> %r1 add %r0, %r1 -> %r0 ld [fp+24] -> %r1 add %r0, %r1 -> %r0 st %r0 -> [fp] ret</pre>	<pre>b: push #20 ld [fp+12] -> %r0 mov #4 -> %r1 mul %r0, %r1 -> %r0 mov #-16 -> %r1 add %r0, %r1 -> %r0 add %r0, %fp -> %r0 ld [r0] -> %r0 ret</pre>
--	---	---

Uninitialized Locals

In `b()`, we allocate the 20 bytes for `p` on the stack, and we know that `p` points to element `p[0]`, which is at address `(fp-16)`. Thus, to access `p[i]` we need to do:

$$\&p[i] = [(fp-16) + 4*[fp+12]]$$



Uninitialized Locals

JASS -- CS360 Assembler

Stack

Step 31

CSR

Registers

r0: 0xffff430
r1: 0xfffffffff0
r2: 0x0
r3: 0x0
r4: 0x0
sp: 0xffff424
fp: 0xffff438
pc: 0x1044

Globals

0xffff424: 0x0
0xffff428: 0x0
0xffff42c: 0x1a
0xffff430: 0xffff448
0xffff434: 0x1074
0xffff438: 0x5
0xffff43c: 0xffff448
0xffff440: 0x1084
0xffff444: 0x2
0xffff448: 0x0

Code

0x1000: push #4
0x1004: ld [fp+12] -> %r0
0x1008: ld [fp+16] -> %r1
0x100c: add %r0, %r1 -> %r0
0x1010: ld [fp+20] -> %r1
0x1014: add %r0, %r1 -> %r0
0x1018: ld [fp+24] -> %r1
0x101c: add %r0, %r1 -> %r0
0x1020: st %r0 -> [fp]
0x1024: ret
0x1028: push #20
0x102c: ld [fp+12] -> %r0
0x1030: mov #4 -> %r1
0x1034: mul %r0, %r1 -> %r0
0x1038: mov #-16 -> %r1
0x103c: add %r0, %r1 -> %r0
0x1040: add %r0, %fp -> %r0
0x1044: ld [r0] -> %r0
0x1048: ret
0x104c: push #4

a

b

main

JASS -- CS360 Assembler

Stack

Step 32

CSR

Registers

r0: 0xffff448
r1: 0xfffffffff0
r2: 0x0
r3: 0x0
r4: 0x0
sp: 0xffff424
fp: 0xffff438
pc: 0x1048

Globals

0xffff424: 0x0
0xffff428: 0x0
0xffff42c: 0x1a
0xffff430: 0xffff448
0xffff434: 0x1074
0xffff438: 0x5
0xffff43c: 0xffff448
0xffff440: 0x1084
0xffff444: 0x2
0xffff448: 0x0

Code

0x1000: push #4
0x1004: ld [fp+12] -> %r0
0x1008: ld [fp+16] -> %r1
0x100c: add %r0, %r1 -> %r0
0x1010: ld [fp+20] -> %r1
0x1014: add %r0, %r1 -> %r0
0x1018: ld [fp+24] -> %r1
0x101c: add %r0, %r1 -> %r0
0x1020: st %r0 -> [fp]
0x1024: ret
0x1028: push #20
0x102c: ld [fp+12] -> %r0
0x1030: mov #4 -> %r1
0x1034: mul %r0, %r1 -> %r0
0x1038: mov #-16 -> %r1
0x103c: add %r0, %r1 -> %r0
0x1040: add %r0, %fp -> %r0
0x1044: ld [r0] -> %r0
0x1048: ret
0x104c: push #4

a

b

main

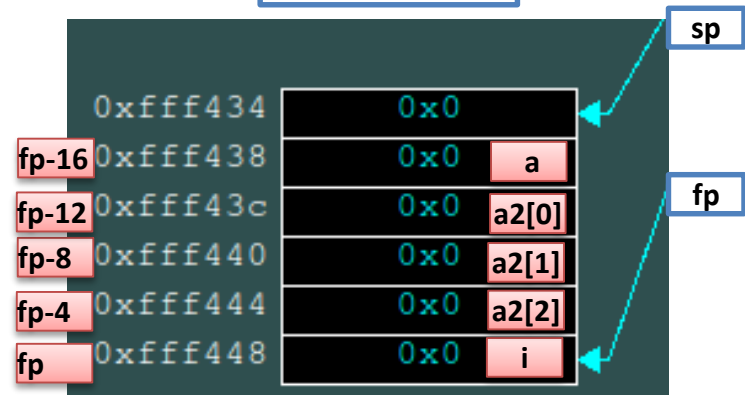
Some more practice

C code

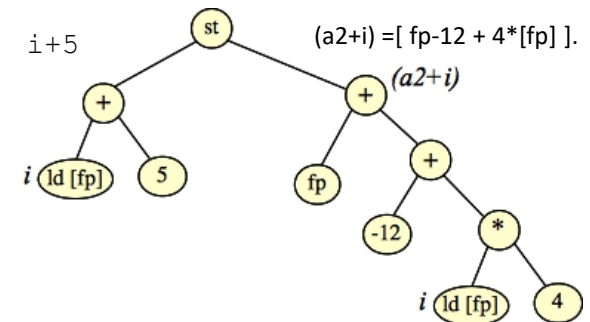
```
main()
{
    int *a, a2[3], i;

    i = 6;
    a = &i;
    a2[1] = i+2;
    *a = 2;
    *(a2+i) = i+5;
}
```

STACK CONTENT



- ☐ &i will be fp.
- ☐ a2 will be (fp-12).
- ☐ a will be [[fp-16]]. You can't do that in assembler. Instead you will load [fp-16] into a register and dereference that register.
- ☐ (a2+i) will be [fp-12 + 4*[fp]]. Remember -- that's pointer arithmetic.



Some more practice

C code

```
main()
{
    int *a, a2[3], i;

    i = 6;
    a = &i;
    a2[1] = i+2;
    *a = 2;
    *(a2+i) = i+5;
}
```

Assembly code

```
main:
    push #20                / Allocate locals and spill r2
    st %r2 -> [sp]--

    mov #6 -> %r0           / i = 6
    st %r0 -> [fp]

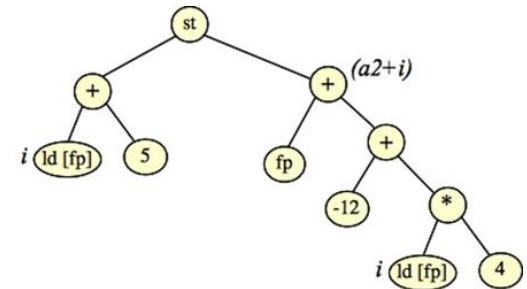
    st %fp -> [fp-16]       / a = &i

    mov #2 -> %r0           / a2[1] = i+2
    ld [fp] -> %r1
    add %r0, %r1 -> %r0
    st %r0 -> [fp-8]

    mov #2 -> %r0           / *a = 2
    ld [fp-16] -> %r1
    st %r0 -> [r1]

    ld [fp] -> %r0          / *(a+i) = i+5
    mov #5 -> %r1
    add %r0, %r1 -> %r0
    ld [fp] -> %r1
    mov #4 -> %r2
    mul %r1, %r2 -> %r1
    mov #-12 -> %r2
    add %r1, %r2 -> %r1
    add %fp, %r1 -> %r1
    st %r0 -> [r1]

    ld ++[sp] -> %r2        / Unspill and exit
    ret
```



Double Indirection

```
int x(int **p, int i, int j)
{
    return p[i+2][j-2];
}
```

```
main()
{
    int a[3], b[3], c[3];
    int *d[3]; int e;

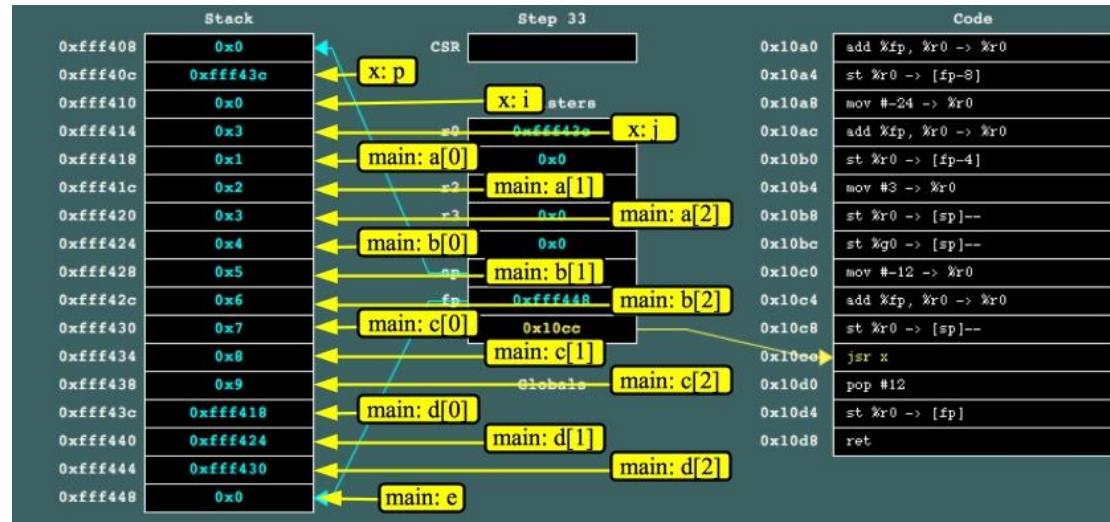
    a[0] = 1; a[1] = 2; a[2] = 3;
    b[0] = 4; b[1] = 5; b[2] = 6;
    c[0] = 7; c[1] = 8; c[2] = 9;
```

```
//d[3][3]
```

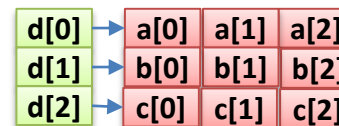
```
d[0] = a; d[1] = b; d[2] = c;
```

```
e = x(d, 0, 3);
```

```
}
```



d[3][3]



d[0][0]=a[0]

d[0][1]=a[1]

...

d[2][2]=c[2]

Double Indirection

```
main()
{
  int a[3], b[3], c[3];
  int *d[3]; int e;
```

```
  a[0] = 1; a[1] = 2; a[2] = 3;
  b[0] = 4; b[1] = 5; b[2] = 6;
  c[0] = 7; c[1] = 8; c[2] = 9;
```

```
  //d[3][3]
```

```
  d[0] = a; d[1] = b; d[2] = c;
```

```
  e = x(d, 0, 3);
```

```
}
```

```
main:
  push #52
  st %g1 -> [fp-48]      / Do a[0] through c[2].
  mov #2 -> %r0
  st %r0 -> [fp-44]
  mov #3 -> %r0
  st %r0 -> [fp-40]
  mov #4 -> %r0
  st %r0 -> [fp-36]
  mov #5 -> %r0
  st %r0 -> [fp-32]
  mov #6 -> %r0
  st %r0 -> [fp-28]
  mov #7 -> %r0
  st %r0 -> [fp-24]
  mov #8 -> %r0
  st %r0 -> [fp-20]
  mov #9 -> %r0
  st %r0 -> [fp-16]

  mov #-48 -> %r0        / d[0] = a
  add %fp, %r0 -> %r0
  st %r0 -> [fp-12]

  mov #-36 -> %r0        / d[1] = b
  add %fp, %r0 -> %r0
  st %r0 -> [fp-8]

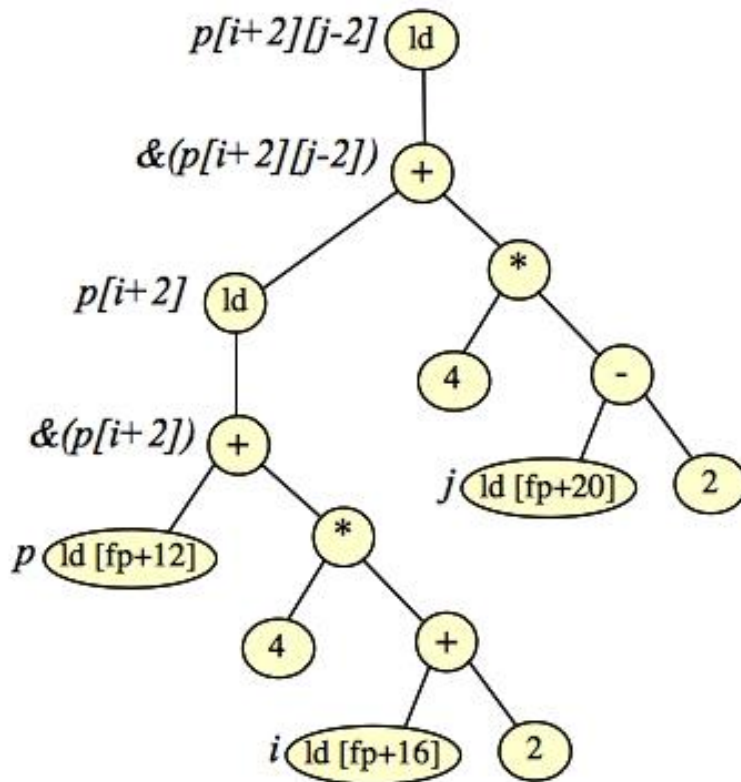
  mov #-24 -> %r0        / d[2] = c
  add %fp, %r0 -> %r0
  st %r0 -> [fp-4]

  mov #3 -> %r0          / Push the arguments in reverse order
  st %r0 -> [sp]--
  st %g0 -> [sp]--
  mov #-12 -> %r0
  add %fp, %r0 -> %r0
  st %r0 -> [sp]--

  jsr x                  / call x and set e
  pop #12
  st %r0 -> [fp]
  ret
```

Double Indirection

```
int x(int **p, int i, int j)
{
    return p[i+2][j-2];
}
```



```
x:
    st %r2 -> [sp]--           / Spill r2

    ld [fp+20] -> %r0           / Do the right part of the tree.
    mov #2 -> %r1
    sub %r0, %r1 -> %r0
    mov #4 -> %r1
    mul %r0, %r1 -> %r0

    mov #2 -> %r1               / Do the left part of the tree
    ld [fp+16] -> %r2
    add %r1, %r2 -> %r1
    mov #4 -> %r2
    mul %r1, %r2 -> %r1
    ld [fp+12] -> %r2
    add %r1, %r2 -> %r1
    ld [%r1] -> %r1

    add %r0, %r1 -> %r0         / Add them up
    ld [%r0] -> %r0

    ld ++[sp] -> %r2           / Unspill r2
    ret
```