Assembler 4

- **Conditionals**
  - if-else
  - while loop
  - for loop
  - recursion

Branch Instructions

- **cmp %r0, %r1** This says to compare the values of the registers r0 and r1, and set the control status register (CSR) to reflect the outcome. The CSR will store whether (r0==r1), (r0 < r1) or (r0 > r1).
- **b l1** This says go (branch) directly to label l1. This sets the pc to l1 rather than (pc+4). Note that you can't "return" from a branch like you can from a "jsr" statement.
- **beq l1** This says that if the CSR denotes that the two compared values are equal, go (set the pc) to label l1. If the two compared values are not equal, the next statement (pc+4) is executed.
- **ble l1** These should be obvious (<=, <, >=, >, !=).
- **blt l1**
- **bge l1**
- **bgt l1**
- **bne l1**

Branch Instructions

```
if (cond) {
   S1
} else {
   S2
}
S3
```

```
set up conditional
   branch on the negation of the conditional to l1
   S1
   b l2
l1:
   S2
l2:
   S3
```

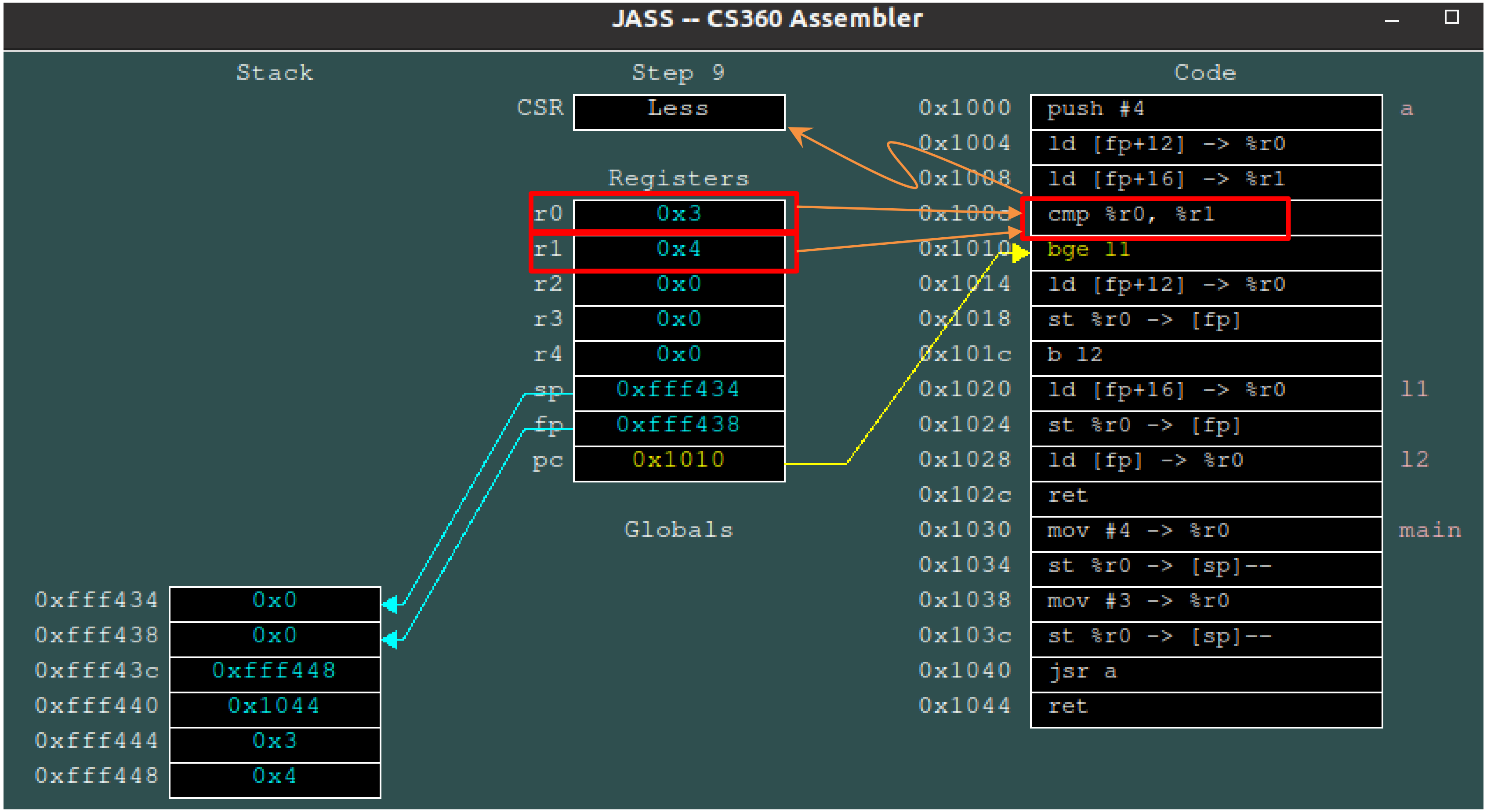
C code

```
int a(int i, int j)
{
  int k;

  if (i < j) {
    k = i;
  } else {
    k = j;
  }
  return k;
}

int main()
{
  return a(3, 4);
}
```

Assembly code

```
a:
  push #4                 / Allocate k

  ld [fp+12] -> %r0       / Compare i & j
  ld [fp+16] -> %r1       / Branch on negation of less-than
  cmp %r0, %r1
  bge l1

  ld [fp+12] -> %r0       / k = i
  st %r0 -> [fp]
  b l2

l1:
  ld [fp+16] -> %r0       / k = j
  st %r0 -> [fp]

l2:
  ld [fp] -> %r0          / return k
  ret

main:
  mov #4 -> %r0
  st %r0 -> [sp]--
  mov #3 -> %r0
  st %r0 -> [sp]--
  jsr a
  ret
```

Branch Instructions



```
JASS -- CS360 Assembler

         Stack                 Step 9                            Code
                         CSR [  Less   ]        0x1000  push #4                  a
                                                0x1004  ld [fp+12] -> %r0
                             Registers          0x1008  ld [fp+16] -> %r1
                         r0  [   0x3   ]        0x100c  cmp %r0, %r1
                         r1  [   0x4   ]        0x1010  bge l1
                         r2  [   0x0   ]        0x1014  ld [fp+12] -> %r0
                         r3  [   0x0   ]        0x1018  st %r0 -> [fp]
                         r4  [   0x0   ]        0x101c  b l2
                         sp  [0xfff434 ]        0x1020  ld [fp+16] -> %r0        l1
                         fp  [0xfff438 ]        0x1024  st %r0 -> [fp]
                         pc  [ 0x1010  ]        0x1028  ld [fp] -> %r0           l2
                                                0x102c  ret
                              Globals           0x1030  mov #4 -> %r0            main
                                                0x1034  st %r0 -> [sp]--
0xfff434 [   0x0    ]                           0x1038  mov #3 -> %r0
0xfff438 [   0x0    ]                           0x103c  st %r0 -> [sp]--
0xfff43c [ 0xfff448 ]                           0x1040  jsr a
0xfff440 [  0x1044  ]                           0x1044  ret
0xfff444 [   0x3    ]
0xfff448 [   0x4    ]
```

Branch Instructions

```
while (cond) {
    S1
}
S2
```

```
l1:
   set up conditional
   branch on the negation of the conditional to l2
   S1
   b l1
l2:
   S2
```

```
for (S1; cond; S2) {
   S3
}
S4
```

```
 S1
   b l2
l1:
   S2
l2:
   set up conditional
   branch on the negation of the conditional to l3
   S3
   b l1
l3:
   S4
```

for-loop example

C code

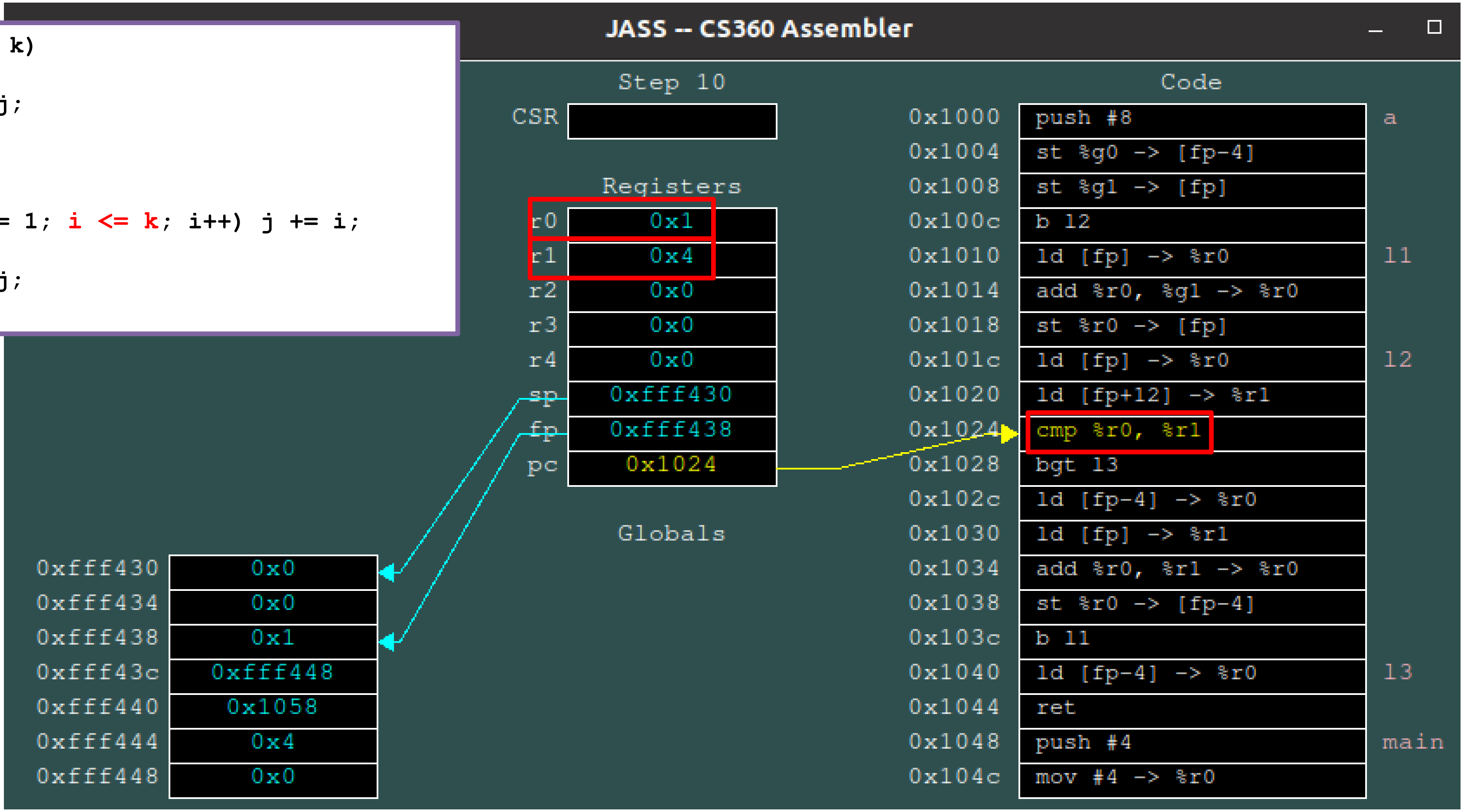
```
int a(int k)
{
  int i, j;

  j = 0;

  for (i = 1; i <= k; i++) j += i;

  return j;
}

int main()
{
  int i;

  i = a(4);
}
```

Assembly code

```
a:
  push #8                          / Allocate i and j on the stack

  st %g0 -> [fp-4]                 / Set j to zero

  st %g1 -> [fp]                   / Initialize the for loop  (S1)
  b l2

l1:

  ld [fp] -> %r0                   / Do i++ (S2)
  add %r0, %g1 -> %r0
  st %r0 -> [fp]

l2:
  ld [fp] -> %r0                   / Perform the test, and
  ld [fp+12] -> %r1                / branch on the negation
  cmp %r0, %r1
  bgt l3

  ld [fp-4] -> %r0                 / Do j += i  (S3)
  ld [fp] -> %r1
  add %r0, %r1 -> %r0
  st %r0 -> [fp-4]
  b l1

l3:
  ld [fp-4] -> %r0                 / return j (S4)
  ret

main:
  push #4

  mov #4 -> %r0
  st %r0 -> [sp]--
  jsr a
  pop #4
  st %r0 -> [fp]
  ret
```

for-loop example

```
int a(int k)
{
  int i, j;

  j = 0;

  for (i = 1; i <= k; i++) j += i;

  return j;
}
```



JASS -- CS360 Assembler

Step 10

CSR

Registers

```
r0   0x1
r1   0x4
r2   0x0
r3   0x0
r4   0x0
sp   0xfff430
fp   0xfff438
pc   0x1024
```

Globals

```
0xfff430   0x0
0xfff434   0x0
0xfff438   0x1
0xfff43c   0xfff448
0xfff440   0x1058
0xfff444   0x4
0xfff448   0x0
```

Code

```
0x1000   push #8                 a
0x1004   st %g0 -> [fp-4]
0x1008   st %g1 -> [fp]
0x100c   b l2
0x1010   ld [fp] -> %r0          l1
0x1014   add %r0, %g1 -> %r0
0x1018   st %r0 -> [fp]
0x101c   ld [fp] -> %r0          l2
0x1020   ld [fp+12] -> %r1
0x1024   cmp %r0, %r1
0x1028   bgt l3
0x102c   ld [fp-4] -> %r0
0x1030   ld [fp] -> %r1
0x1034   add %r0, %r1 -> %r0
0x1038   st %r0 -> [fp-4]
0x103c   b l1
0x1040   ld [fp-4] -> %r0        l3
0x1044   ret
0x1048   push #4                 main
0x104c   mov #4 -> %r0
```

Recursion example

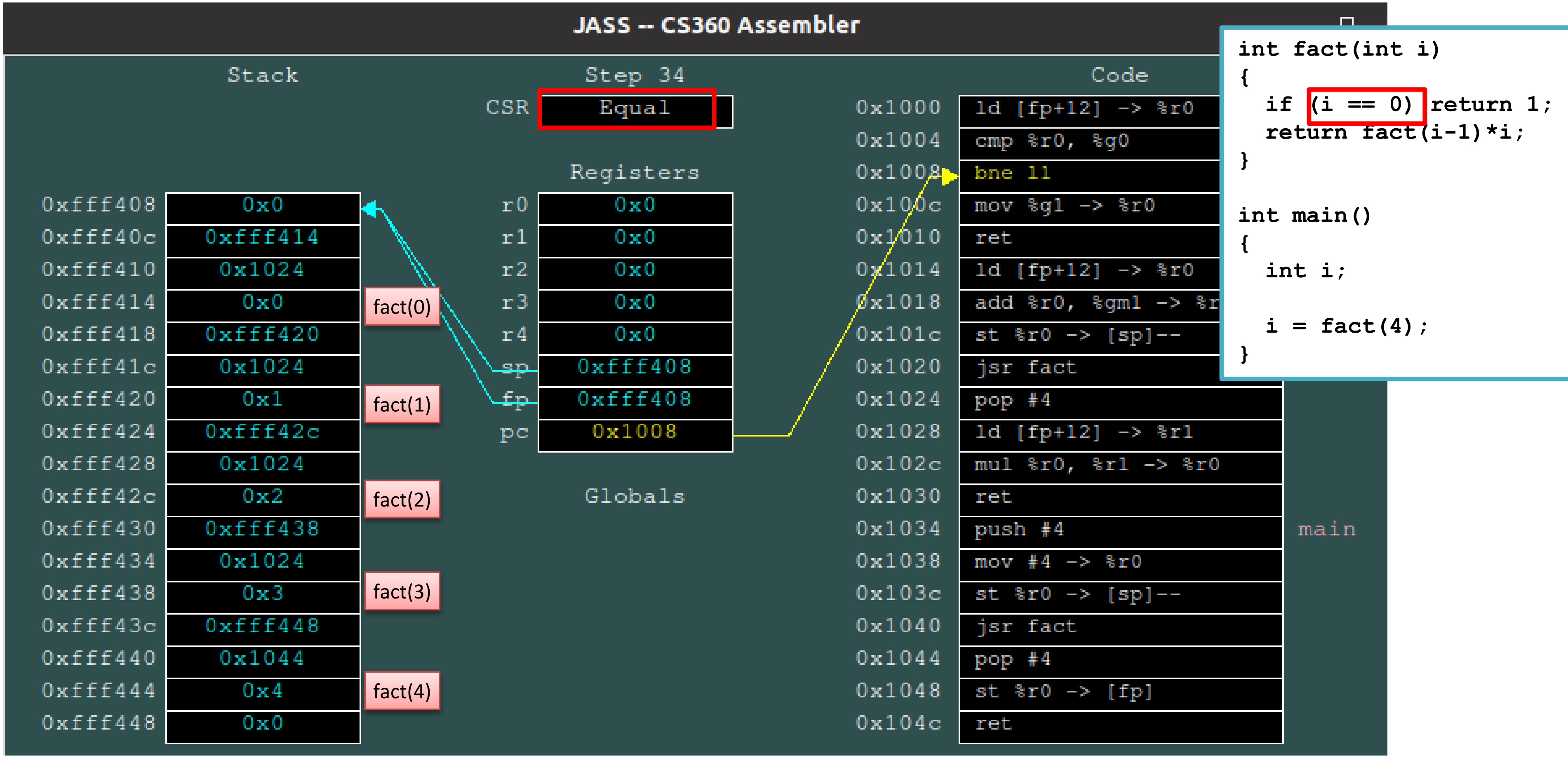
C code

```
int fact(int i)
{
  if (i == 0) return 1;
  return fact(i-1)*i;
}

int main()
{
  int i;

  i = fact(4);
}
```

Assembly code

```
fact:
    ld [fp+12] -> %r0          / do the if statement
    cmp %r0, %g0
    bne l1
    mov %g1 -> %r0
    ret

l1:
    ld [fp+12] -> %r0          / push i-1 on the stack
    add %r0, %gm1 -> %r0
    st %r0 -> [sp]--
    jsr fact                   / recursive call to fact
    pop #4                     / pop the argument off the
stack

    ld [fp+12] -> %r1          / multiply fact(i-1)*i
    mul %r0, %r1 -> %r0
    ret

main:
    push #4
    mov #4 -> %r0
    st %r0 -> [sp]--
    jsr fact
    pop #4
    st %r0 -> [fp]
    ret
```

Recursion example



JASS -- CS360 Assembler

Step 34

CSR: Equal

Stack

0xfff408	0x0	
0xfff40c	0xfff414	
0xfff410	0x1024	
0xfff414	0x0	fact(0)
0xfff418	0xfff420	
0xfff41c	0x1024	
0xfff420	0x1	fact(1)
0xfff424	0xfff42c	
0xfff428	0x1024	
0xfff42c	0x2	fact(2)
0xfff430	0xfff438	
0xfff434	0x1024	
0xfff438	0x3	fact(3)
0xfff43c	0xfff448	
0xfff440	0x1044	
0xfff444	0x4	fact(4)
0xfff448	0x0	

Registers

r0	0x0
r1	0x0
r2	0x0
r3	0x0
r4	0x0
sp	0xfff408
fp	0xfff408
pc	0x1008

Globals

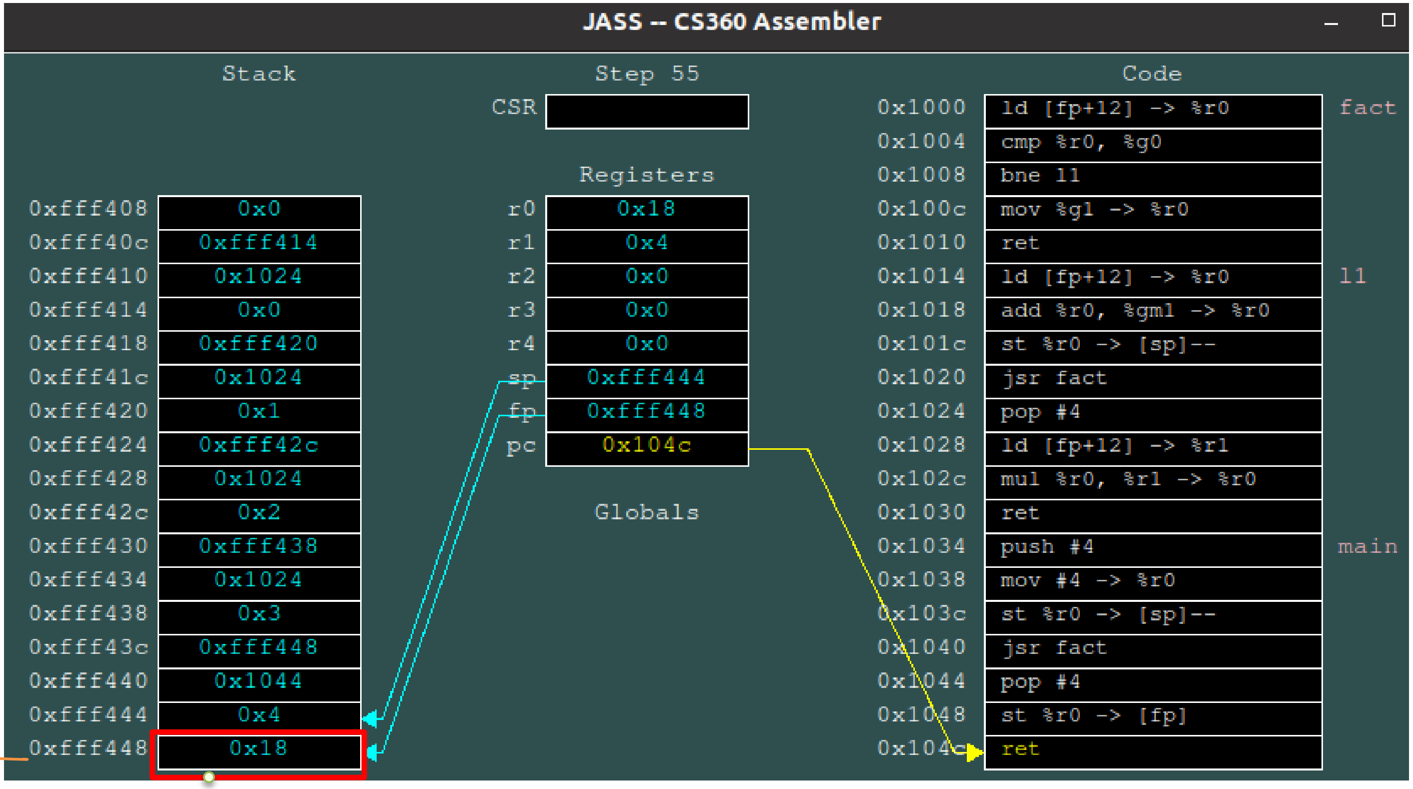
Code

0x1000	ld [fp+12] -> %r0	
0x1004	cmp %r0, %g0	
0x1008	bne l1	
0x100c	mov %g1 -> %r0	
0x1010	ret	
0x1014	ld [fp+12] -> %r0	
0x1018	add %r0, %gm1 -> %r	
0x101c	st %r0 -> [sp]--	
0x1020	jsr fact	
0x1024	pop #4	
0x1028	ld [fp+12] -> %r1	
0x102c	mul %r0, %r1 -> %r0	
0x1030	ret	
0x1034	push #4	main
0x1038	mov #4 -> %r0	
0x103c	st %r0 -> [sp]--	
0x1040	jsr fact	
0x1044	pop #4	
0x1048	st %r0 -> [fp]	
0x104c	ret	

```
int fact(int i)
{
  if (i == 0) return 1;
  return fact(i-1)*i;
}

int main()
{
  int i;

  i = fact(4);
}
```

Recursion example

```
int fact(int i)
{
  if (i == 0) return 1;
  return fact(i-1)*i;
}

int main()
{
  int i;

  i = fact(4);
}
```



```
                              JASS -- CS360 Assembler

           Stack                     Step 55                       Code
                              CSR                      0x1000  ld [fp+12] -> %r0      fact
                                                       0x1004  cmp %r0, %g0
                                    Registers          0x1008  bne l1
0xfff408   0x0                r0    0x18               0x100c  mov %g1 -> %r0
0xfff40c   0xfff414           r1    0x4                0x1010  ret
0xfff410   0x1024             r2    0x0                0x1014  ld [fp+12] -> %r0      l1
0xfff414   0x0                r3    0x0                0x1018  add %r0, %gm1 -> %r0
0xfff418   0xfff420           r4    0x0                0x101c  st %r0 -> [sp]--
0xfff41c   0x1024             sp    0xfff444           0x1020  jsr fact
0xfff420   0x1                fp    0xfff448           0x1024  pop #4
0xfff424   0xfff42c           pc    0x104c             0x1028  ld [fp+12] -> %r1
0xfff428   0x1024                                      0x102c  mul %r0, %r1 -> %r0
0xfff42c   0x2                      Globals            0x1030  ret
0xfff430   0xfff438                                    0x1034  push #4                main
0xfff434   0x1024                                      0x1038  mov #4 -> %r0
0xfff438   0x3                                         0x103c  st %r0 -> [sp]--
0xfff43c   0xfff448                                    0x1040  jsr fact
0xfff440   0x1044                                      0x1044  pop #4
0xfff444   0x4                                         0x1048  st %r0 -> [fp]
0xfff448   0x18                                        0x104c  ret
```

Result of the factorial.

$4!=(24)_{10}=(18)_{16}$