# Prelim Interlude

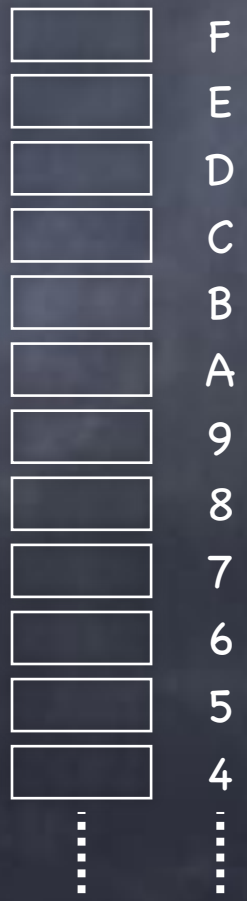
- 32bit architecture, byte addressable. The stack grows down from high to low addresses

Addresses from
0x0 to 0xFFFF FFFF

- 32bit architecture, byte addressable. The stack grows down from high to low addresses

What is the address of the next to last word?

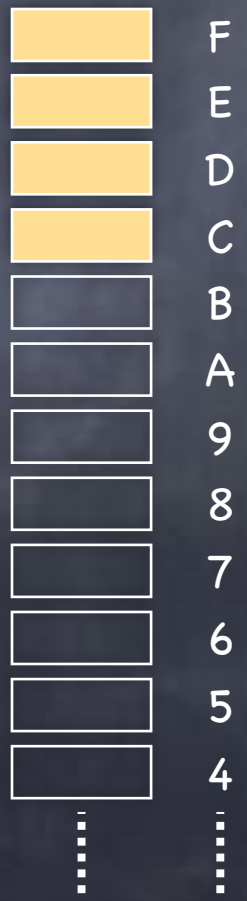
Addresses from 0x0 to 0xFFFF FFFF



- 32bit architecture, byte addressable. The stack grows down from high to low addresses

What is the address of the next to last word?

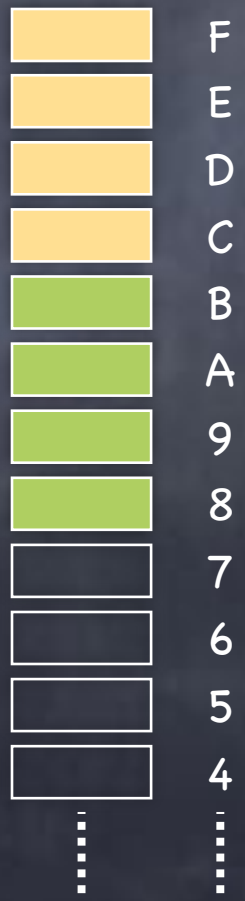
Addresses from 0x0 to 0xFFFF FFFF



- 32bit architecture, byte addressable. The stack grows down from high to low addresses

What is the address of the next to last word?

Addresses from 0x0 to 0xFFFF FFFF




0xFFFF FFF8

- 32bit architecture, byte addressable. The stack grows down from high to low addresses

If the stack pointer is at 0xFFFE 8D70 and three words are pushed onto the stack, what is the new value?

3 words = 12 bytes

downward

0xFFFE 8D70 - 0xB = 0xFFFE 8D64

- 32bit architecture, byte addressable. The stack grows down from high to low addresses

User process occupies bottom half of the 32-bit address space (i.e., the lower addresses), while the kernel occupies the top half of the same address space (i.e., the higher addresses)

What is the address where the kernel starts (i.e. the lowest address in the kernel)?

0xFFFF FFFF / 2

Right shift by one position

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

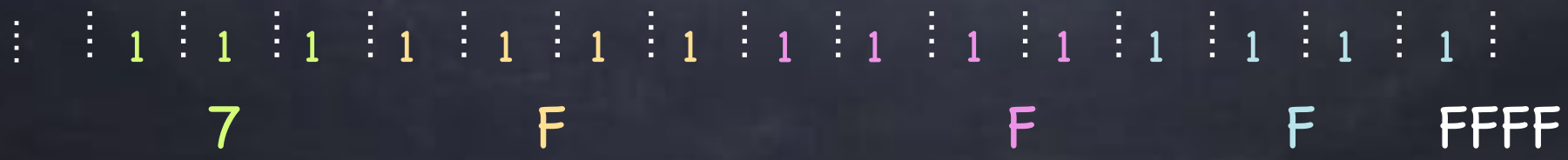
- 32bit architecture, byte addressable. The stack grows down from high to low addresses

User process occupies bottom half of the 32-bit address space (i.e., the lower addresses), while the kernel occupies the top half of the same address space (i.e., the higher addresses)

What is the address where the kernel starts (i.e. the lowest address in the kernel)?

0xFFFF FFFF / 2

Right shift by one position



- 32bit architecture, byte addressable. The stack grows down from high to low addresses

User process occupies bottom half of the 32-bit address space (i.e., the lower addresses), while the kernel occupies the top half of the same address space (i.e., the higher addresses)

What is the address where the kernel starts (i.e. the lowest address in the kernel)?

0x7FFF FFFF + 1 = 0x8000 0000

- 32bit architecture, byte addressable. The stack grows down from high to low addresses

User process occupies bottom half of the 32-bit address space (i.e., the lower addresses), while the kernel occupies the top half of the same address space (i.e., the higher addresses)

What is the address of the last byte of a user process (i.e. the highest user space address)?

The table on the right shows the interrupt (trap) vector.

Suppose the CPU is executing in user space. The program counter is **0x00006FC8**, the user stack pointer is 0x7FF277E0, and the kernel stack pointer is **0xFFFA37C0**. Now a disk interrupt occurs, pushing the user's PC, SP, and PSW onto the kernel stack. The interrupt handler pushes 5 general purpose registers onto the stack. Each push instruction occupies 4 bytes.

Answer the following questions about the CPU's state at this moment:

| Index | Address | Type |
|---|---|---|
| 0 | 0x806ECEB0 | System Call |
| 1 | 0x8050D330 | Divide-by-Zero |
| 2 | 0x80248280 | Page Fault |
| 3 | 0x80001079 | Disk Interrupt |
| 4 | 0x80520420 | Clock Interrupt |

What is the value of the user stack pointer?

0x7FF277E0

The table on the right shows the interrupt (trap) vector.

Suppose the CPU is executing in user space. The program counter is **0x00006FC8**, the user stack pointer is **0x7FF277E0**, and the kernel stack pointer is 0xFFFA37C0. Now a disk interrupt occurs, pushing the user's PC, SP, and PSW onto the kernel stack. The interrupt handler pushes 5 general purpose registers onto the stack. Each push instruction occupies 4 bytes.

Answer the following questions about the CPU's state at this moment:

| Index | Address | Type |
|---|---|---|
| 0 | 0x806ECEB0 | System Call |
| 1 | 0x8050D330 | Divide-by-Zero |
| 2 | 0x80248280 | Page Fault |
| 3 | 0x80001079 | Disk Interrupt |
| 4 | 0x80520420 | Clock Interrupt |

What is the value of the kernel stack pointer?

KSP starts at 0xFFFA37C0

Push { User's PC, SP, PSW
5 general purpose registers

8 words — downwards

0xFFFA37C0 - 0x20 = 0xFFFA37A0

The table on the right shows the interrupt (trap) vector.

Suppose the CPU is executing in user space. The program counter is **0x00006FC8**, the user stack pointer is **0x7FF277E0**, and the kernel stack pointer is **0xFFFA37C0**. Now a disk interrupt occurs, pushing the user's PC, SP, and PSW onto the kernel stack. The interrupt handler pushes 5 general purpose registers onto the stack. Each push instruction occupies 4 bytes.

Answer the following questions about the CPU's state at this moment:

| Index | Address | Type |
|---|---|---|
| 0 | 0x806ECEB0 | System Call |
| 1 | 0x8050D330 | Divide-by-Zero |
| 2 | 0x80248280 | Page Fault |
| 3 | 0x80001079 | Disk Interrupt |
| 4 | 0x8052C420 | Clock Interrupt |

What is the value of value of the PC?

PC starts at 0x80001079

5 push instructions, each 4 bytes = 20 bytes

0x80001079 + 0x14 = 0x8000108D

The table on the right shows the interrupt (trap) vector.

Suppose the CPU is executing in user space. The program counter is **0x00006FC8**, the user stack pointer is **0x7FF277E0**, and the kernel stack pointer is **0xFFFA37C0**. Now a disk interrupt occurs, pushing the user's PC, SP, and PSW onto the kernel stack. The interrupt handler pushes 5 general purpose registers onto the stack. Each push instruction occupies 4 bytes.

Answer the following questions about the CPU's state at this moment:

| Index | Address | Type |
|---|---|---|
| 0 | 0x806ECEB0 | System Call |
| 1 | 0x8050D330 | Divide-by-Zero |
| 2 | 0x80248280 | Page Fault |
| 3 | 0x80001079 | Disk Interrupt |
| 4 | 0x80520420 | Clock Interrupt |

Now, suppose the disk interrupt handler has completed and has just executed the return-from-interrupt instruction. Answer the following questions about the CPU state at that time.

What is the value of the user SP?

0x7FF277E0

The table on the right shows the interrupt (trap) vector.

Suppose the CPU is executing in user space. The program counter is **0x00006FC8**, the user stack pointer is **0x7FF277E0**, and the kernel stack pointer is 0xFFFA37C0. Now a disk interrupt occurs, pushing the user's PC, SP, and PSW onto the kernel stack. The interrupt handler pushes 5 general purpose registers onto the stack. Each push instruction occupies 4 bytes.

Answer the following questions about the CPU's state at this moment:

| Index | Address | Type |
|---|---|---|
| 0 | 0x806ECEB0 | System Call |
| 1 | 0x8050D330 | Divide-by-Zero |
| 2 | 0x80248280 | Page Fault |
| 3 | 0x80001079 | Disk Interrupt |
| 4 | 0x80520420 | Clock Interrupt |

Now, suppose the disk interrupt handler has completed and has just executed the return-from-interrupt instruction. Answer the following questions about the CPU state at that time.

What is the value of the kernel SP?

0xFFFA37C0

The table on the right shows the interrupt (trap) vector.

Suppose the CPU is executing in user space. The program counter is 0x00006FC8, the user stack pointer is **0x7FF277E0**, and the kernel stack pointer is **0xFFFA37C0**. Now a disk interrupt occurs, pushing the user's PC, SP, and PSW onto the kernel stack. The interrupt handler pushes 5 general purpose registers onto the stack. Each push instruction occupies 4 bytes.

Answer the following questions about the CPU's state at this moment:

| Index | Address | Type |
|---|---|---|
| 0 | 0x806ECEB0 | System Call |
| 1 | 0x8050D330 | Divide-by-Zero |
| 2 | 0x80248280 | Page Fault |
| 3 | 0x80001079 | Disk Interrupt |
| 4 | 0x80520420 | Clock Interrupt |

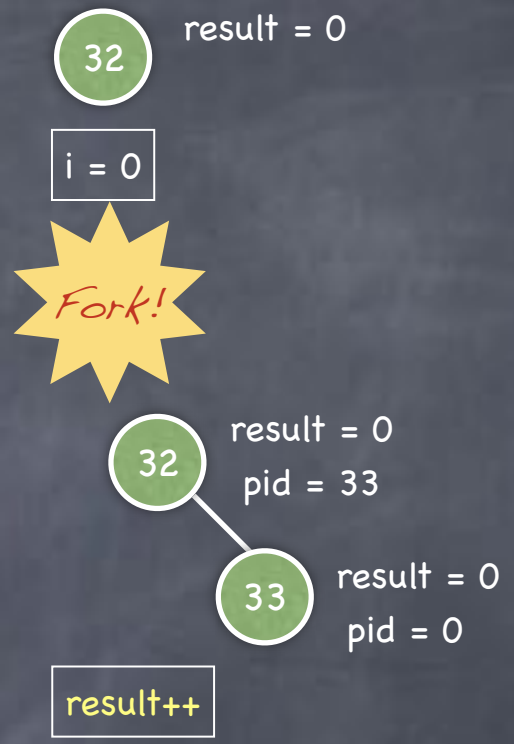
Now, suppose the disk interrupt handler has completed and has just executed the return-from-interrupt instruction. Answer the following questions about the CPU state at that time.

What is the value of the PC?

0x00006FC8

```c
#include <stdio.h>  /* declares printf() */
#include <unistd.h> /* declares fork() */


int main() {
    int i;
    int pid;
    int result = 0;
    for (i=0; i<2; i++) {
        pid = fork();
        result ++;
        printf ("result = %d\n", result);
    }
    if (pid == 0) {
        printf ("result = %d\n", result);
    }
    return 0;
}
```




How many times will the value of result be printed?

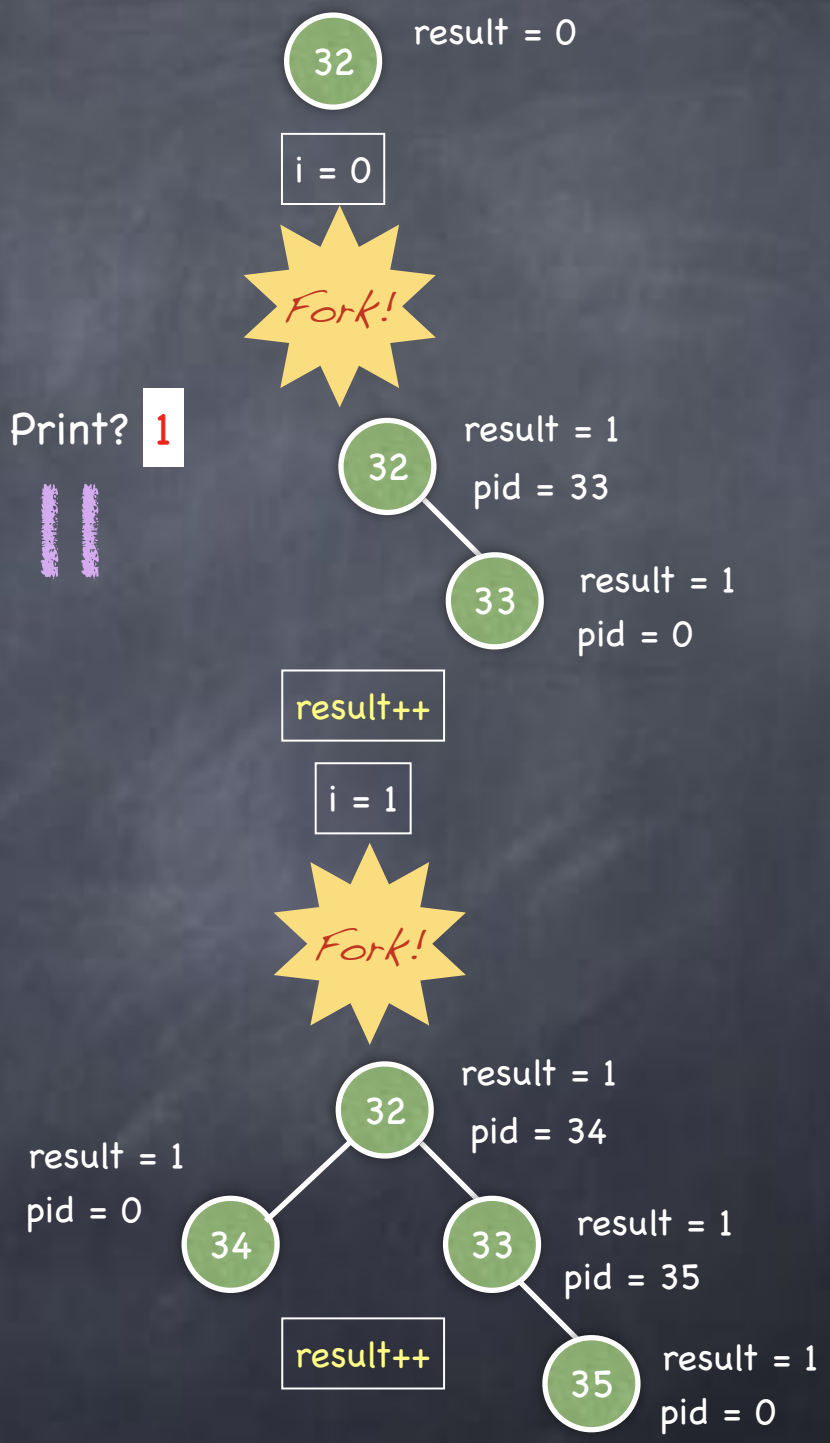
First value(s)? Last value(s)?

```c
#include <stdio.h>  /* declares printf() */
#include <unistd.h> /* declares fork() */


int main() {
    int i;
    int pid;
    int result = 0;
    for (i=0; i<2; i++) {
        pid = fork();
        result ++;
        printf ("result = %d\n", result);
    }
    if (pid == 0) {
        printf ("result = %d\n", result);
    }
    return 0;
}
```

How many times will the value of result be printed?

First value(s)? Last value(s)?

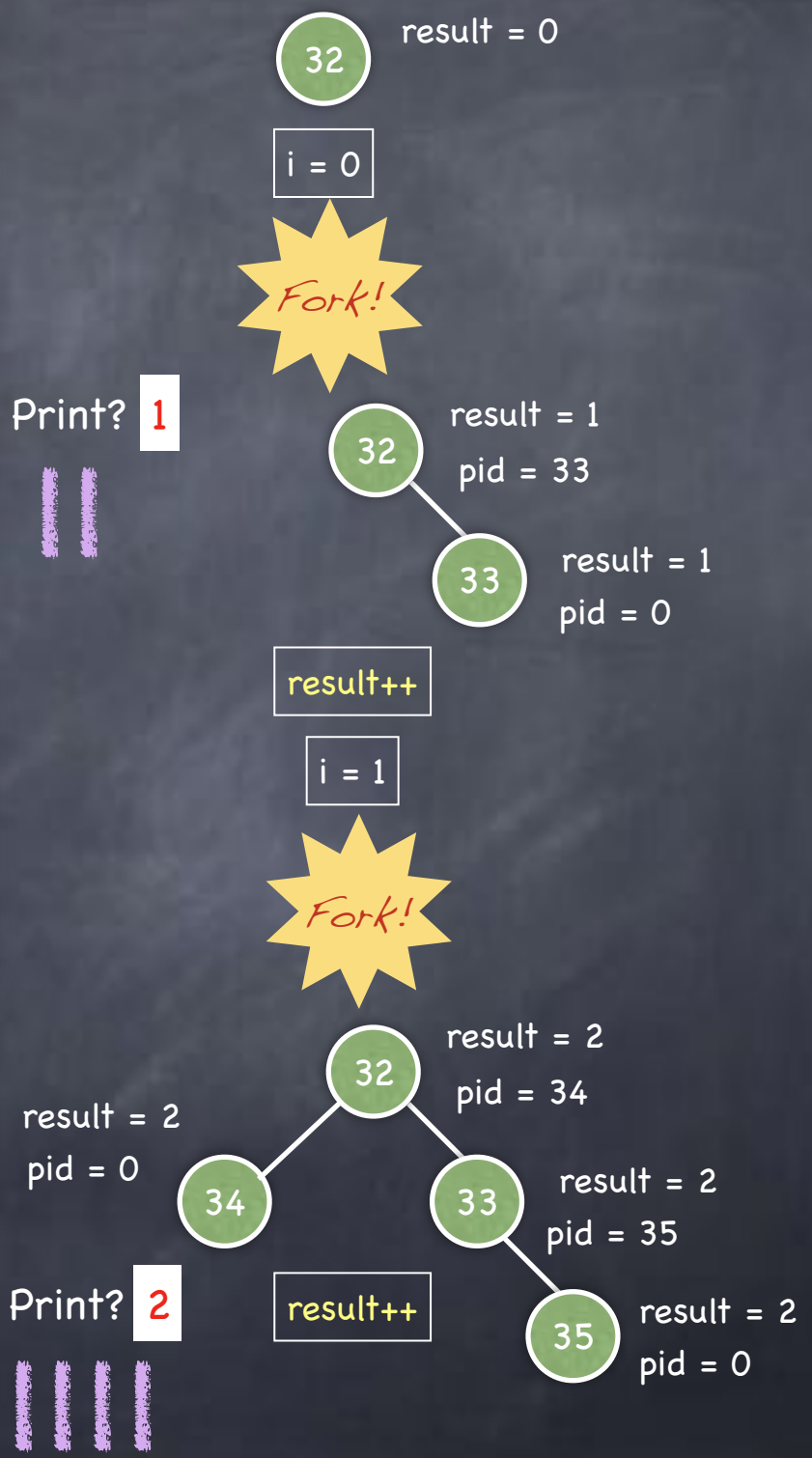


```c
#include <stdio.h>  /* declares printf() */
#include <unistd.h> /* declares fork() */


int main() {
    int i;
    int pid;
    int result = 0;
    for (i=0; i<2; i++) {
        pid = fork();
        result ++;
        printf ("result = %d\n", result);
    }
    if (pid == 0) {
        printf ("result = %d\n", result);
    }
    return 0;
}
```

How many times will the value of result be printed?

First value(s)? Last value(s)?



```c
#include <stdio.h>  /* declares printf() */
#include <unistd.h> /* declares fork() */


int main() {
    int i;
    int pid;
    int result = 0;
    for (i=0; i<2; i++) {
        pid = fork();
        result ++;
        printf ("result = %d\n", result);
    }
    if (pid == 0) {
        printf ("result = %d\n", result);
    }
    return 0;
}
```

How many times will the value of result be printed?

First value(s)? Last value(s)?