# Assembly Programming for the


Chris Patuzzo

# About me

London Computation Club

Software Developer

Hard search problems

# This talk

How to get started

Basics of CPU architecture

Solving Hamiltonian path

# By the end

Access to a powerful tool

Know more about your computer

Where to go next

# Access to a powerful tool

Access to a powerful tool

216,000,000 iterations/second

```ruby
Benchmark.realtime { 1.upto(216_000_000) {} }
#=> 7.0587
```

# What is this?

# “System on a chip”

CPU


“Secure Enclave”

GPU

“Neural Engine”

Memory

# “System on a chip”

CPU


“Secure Enclave”

GPU

“Neural Engine”

Memory

# CPU

Runs ARM64 (v8.5-A)

RISC architecture

Low power, less heat

https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools

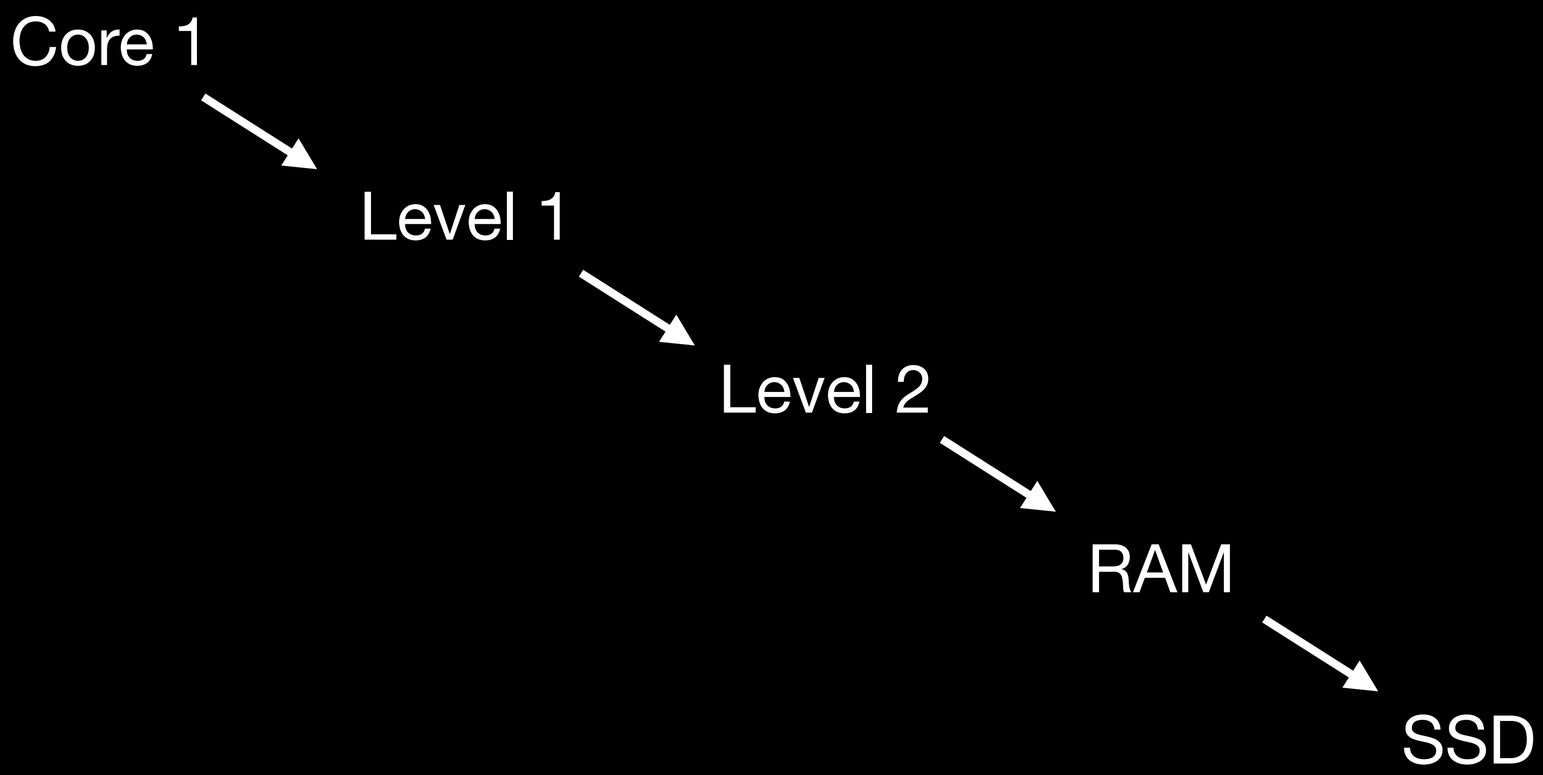
# CPU

4 “Firestorm” cores

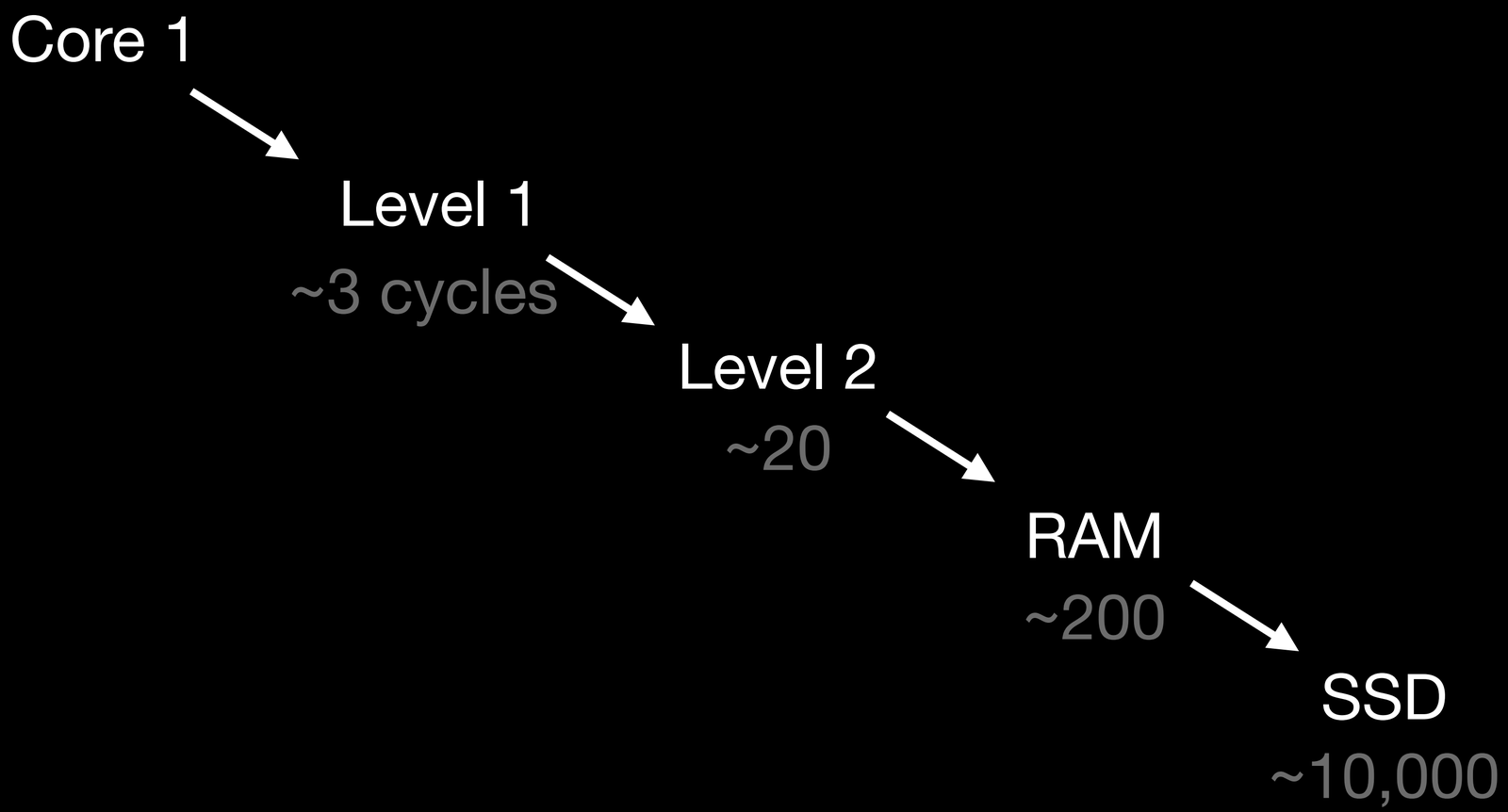
4 “Icestorm” cores

(different speeds / cache)

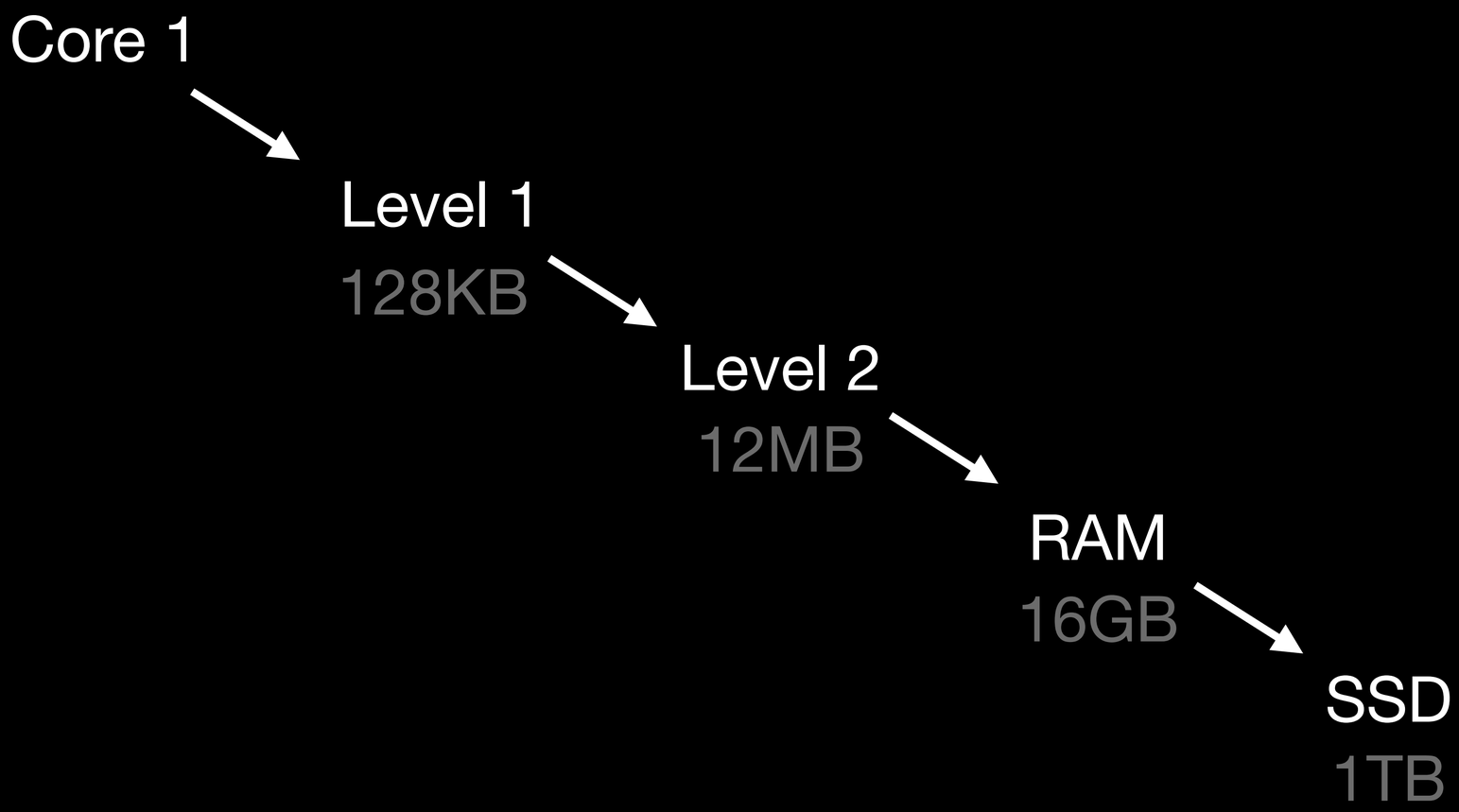
# What is cache?



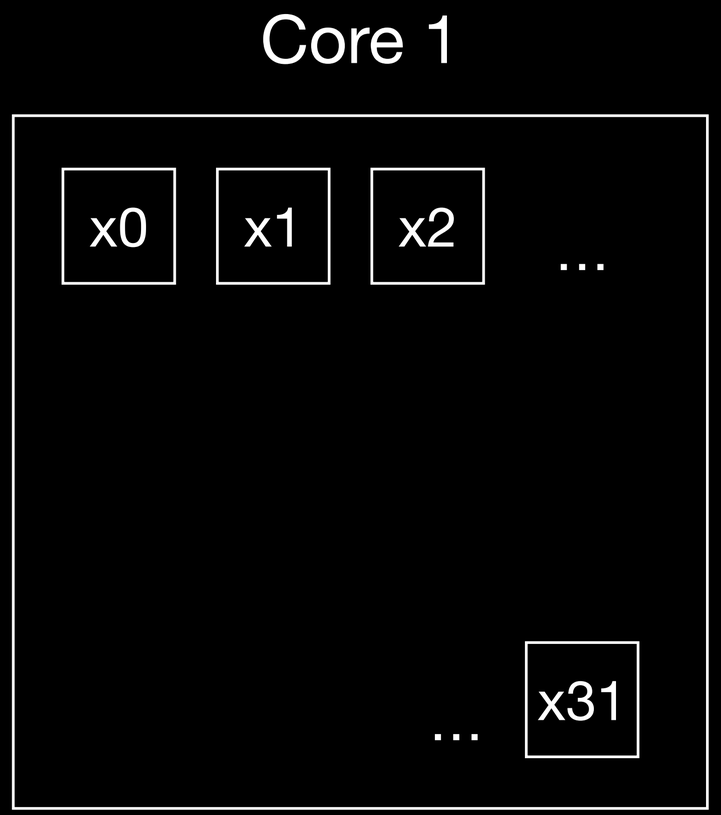
# What is cache?



# What is cache?



# Registers




32 registers (1 cycle)

64 bits each

Some are special

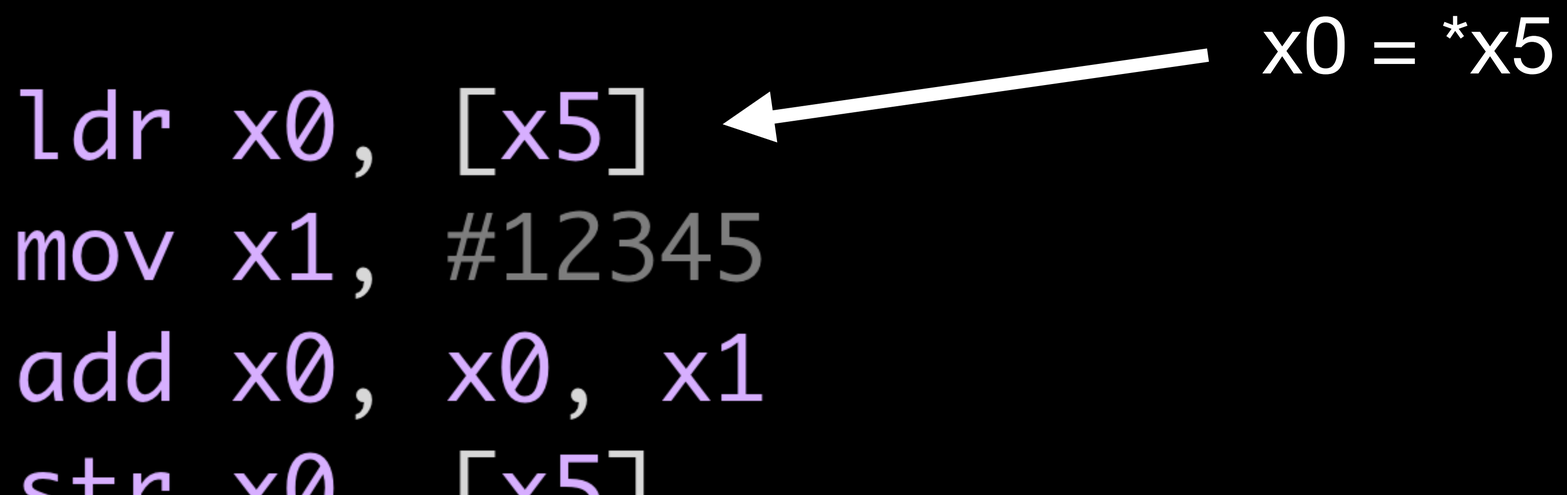
# Load Store Architecture

Load from memory

Calculate something

Store into memory

```
ldr x0, [x5]
mov x1, #12345
add x0, x0, x1
str x0, [x5]
```


x0 = *x5
ldr x0, [x5]
mov x1, #12345
add x0, x0, x1
str x0, [x5]

```
ldr x0, [x5]
mov x1, #12345
add x0, x0, x1
str x0, [x5]
```

```
ldr x0, [x5]
mov x1, #12345
add x0, x0, x1
str x0, [x5]
```

```
ldr x0, [x5]
mov x1, #12345
add x0, x0, x1
str x0, [x5]
```

*x5 = x0

# Service Calls

Supervising process

Ask it to do something

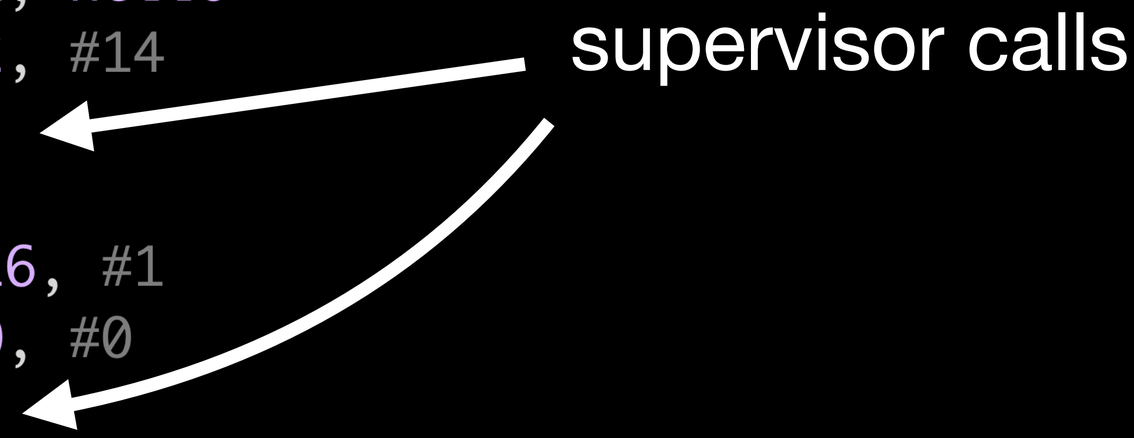
e.g. print, exit

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

} boilerplate

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```



```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

```
.global _main
.align 2

_main:
  mov x16, #4
  mov x0, #1
  adr x1, hello
  mov x2, #14
  svc 0

  mov x16, #1
  mov x0, #0
  svc 0

hello: .ascii "Hello, world!\n"
```

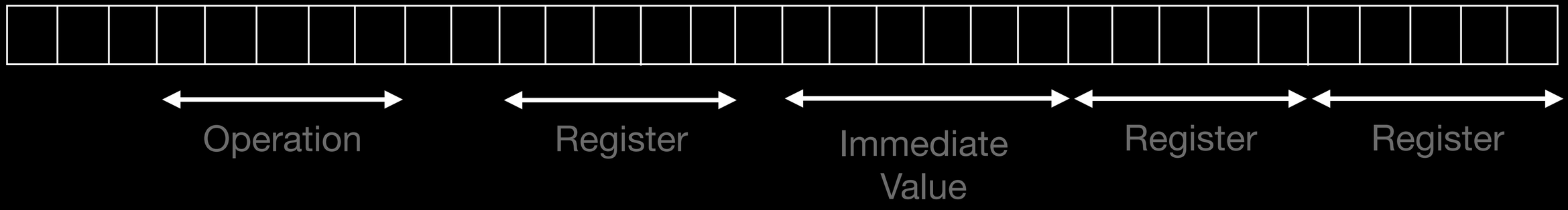
# Assemble and Link

```
#!/bin/bash

sdk=`xcrun -sdk macosx --show-sdk-path`

as -arch arm64 hello.s -o hello.o
ld -arch arm64 hello.o -o hello -lSystem -syslibroot $sdk
```

```
$ ./hello
Hello, world!
```

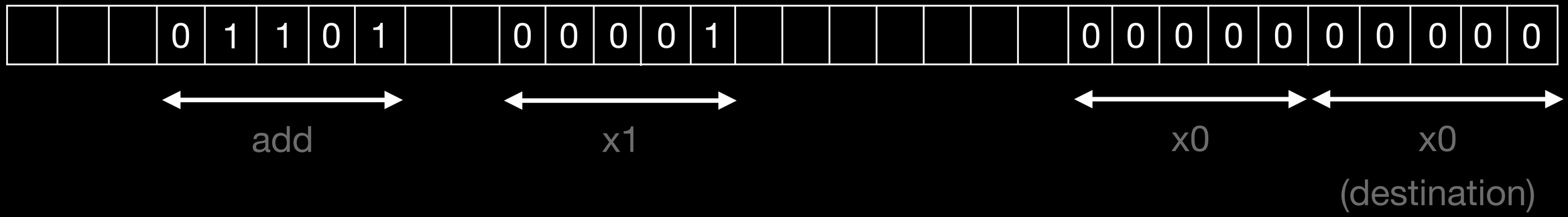
# Instructions

32 bits wide




(some bits omitted)

```
add x0, x0, x1
```

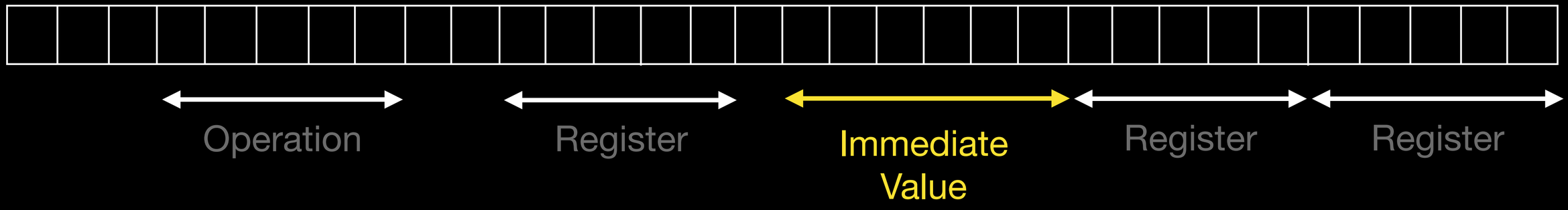



(some bits omitted)

```
mov x1, #12345
add x0, x0, x1
```

versus.

```
add x0, x0, #12345
```


Operation
Register
Immediate
Value
Register
Register

# What next?

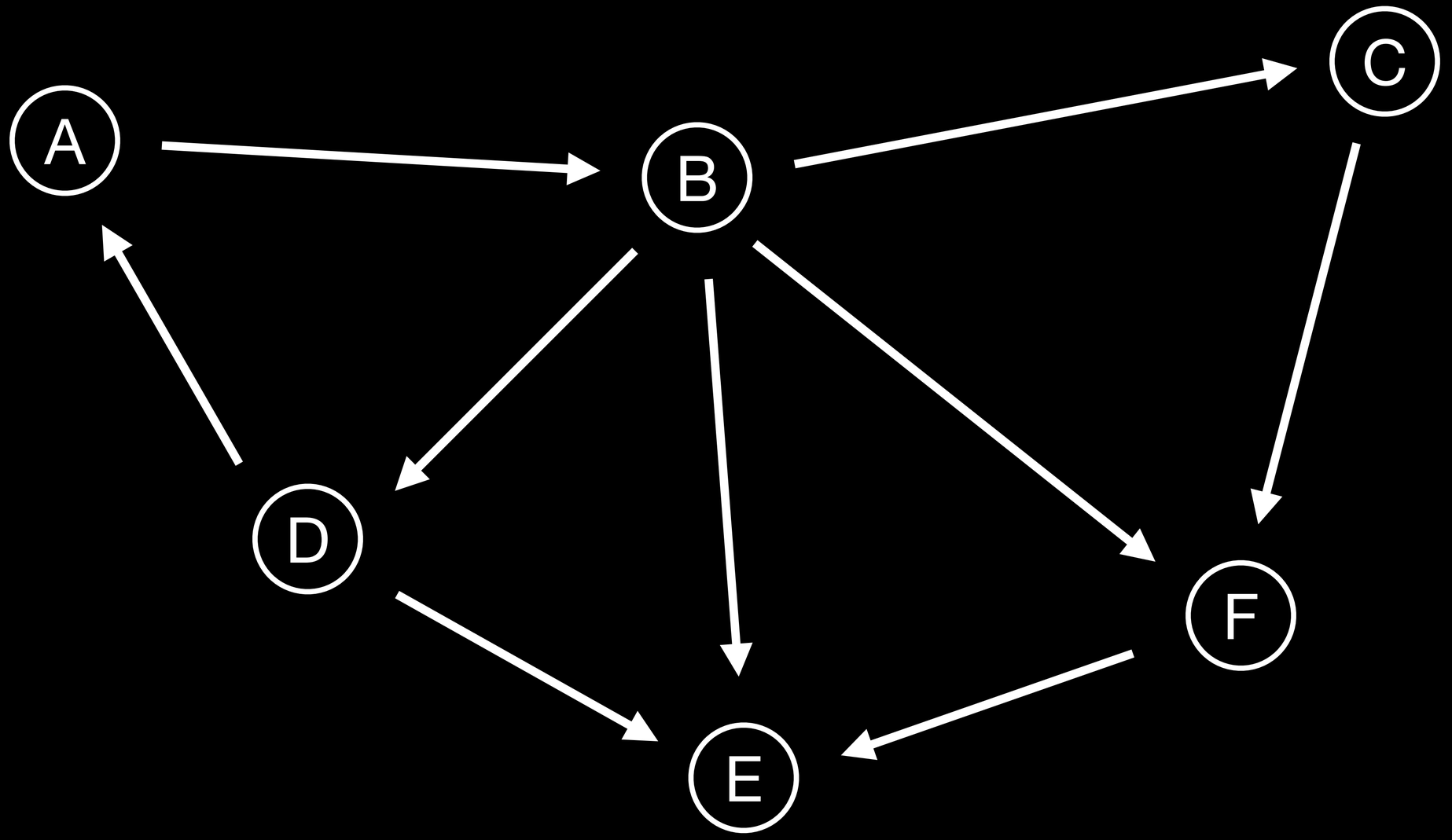
Branching?

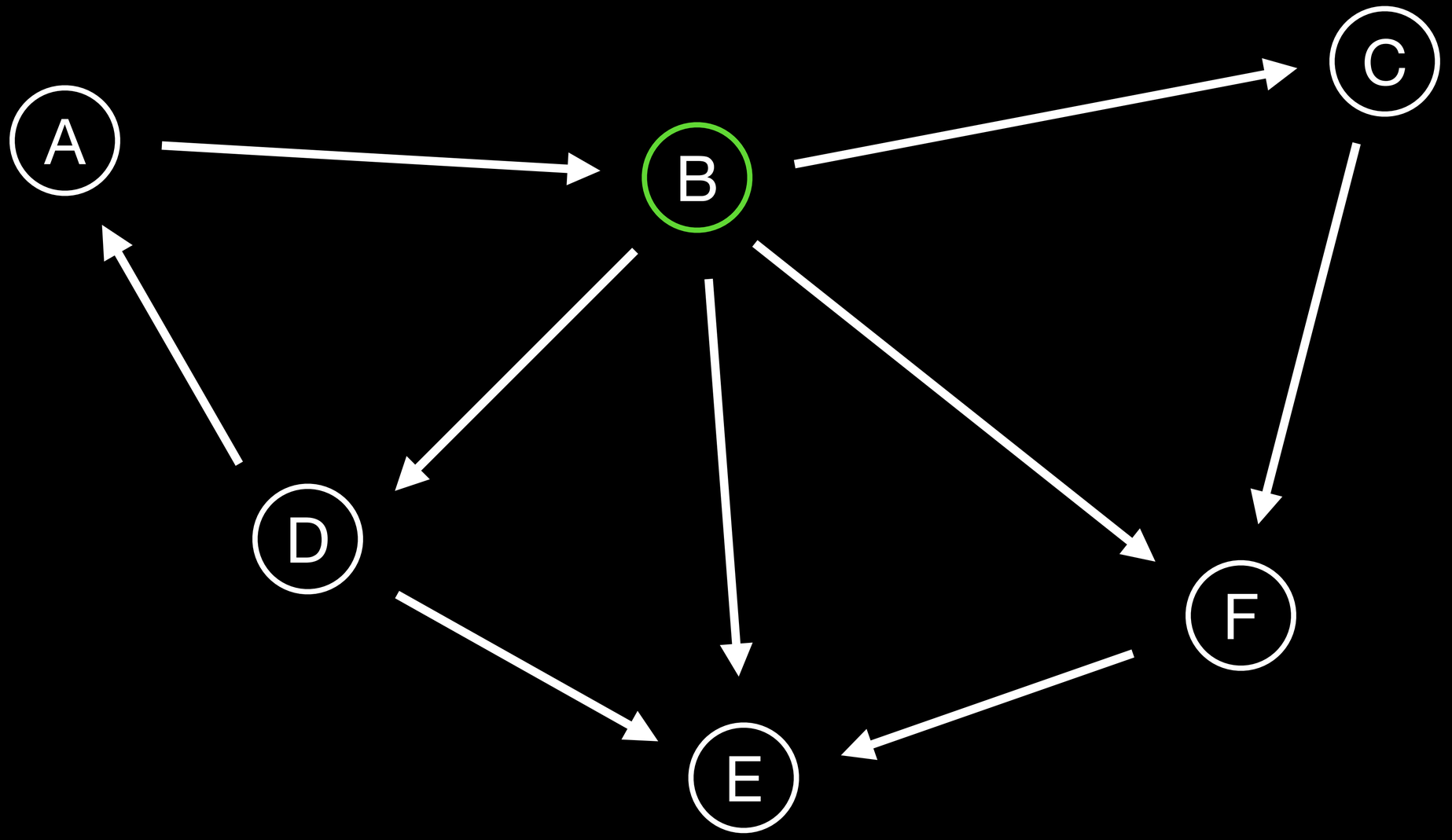
Logical operations?

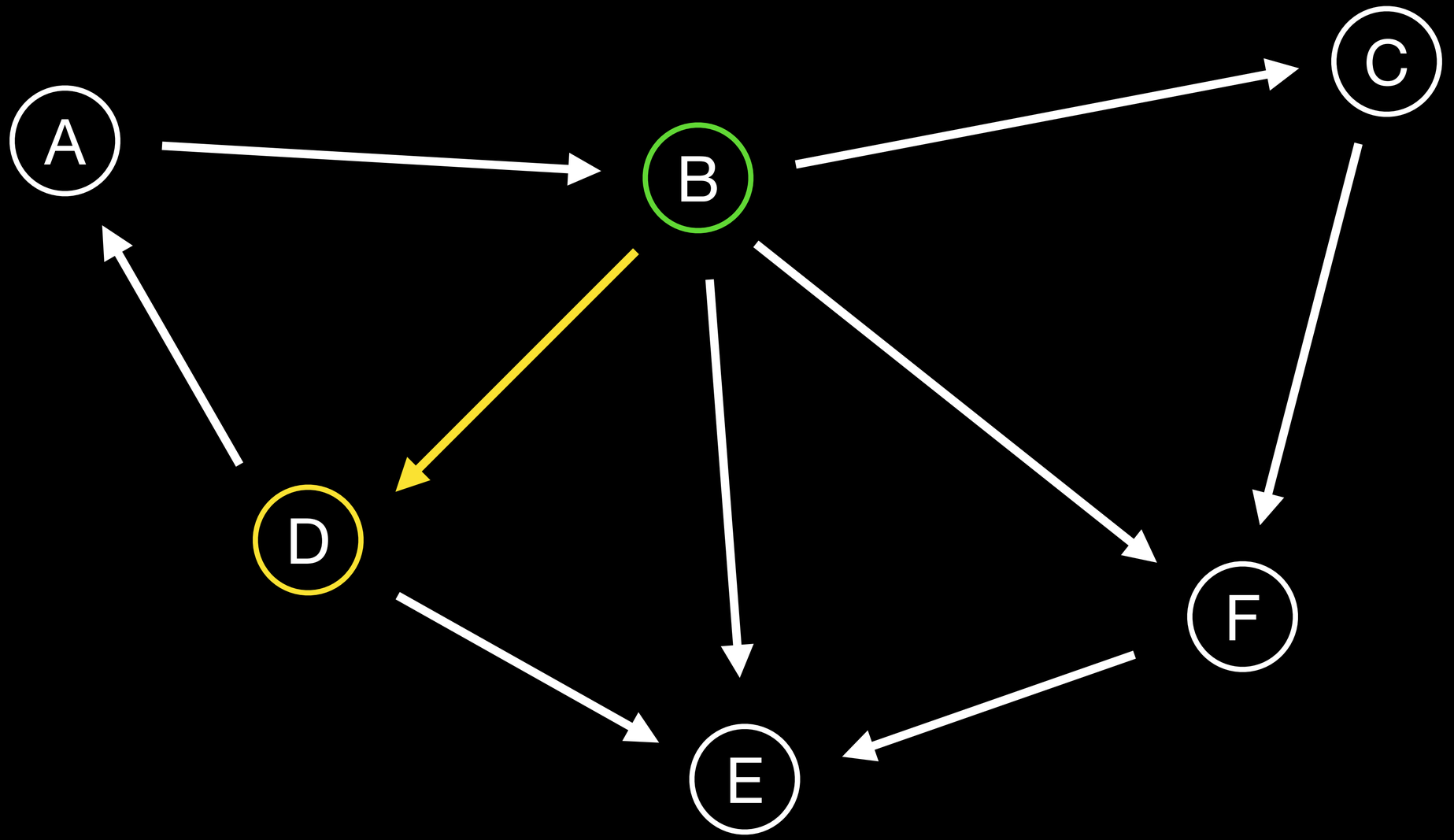
Memory operations?

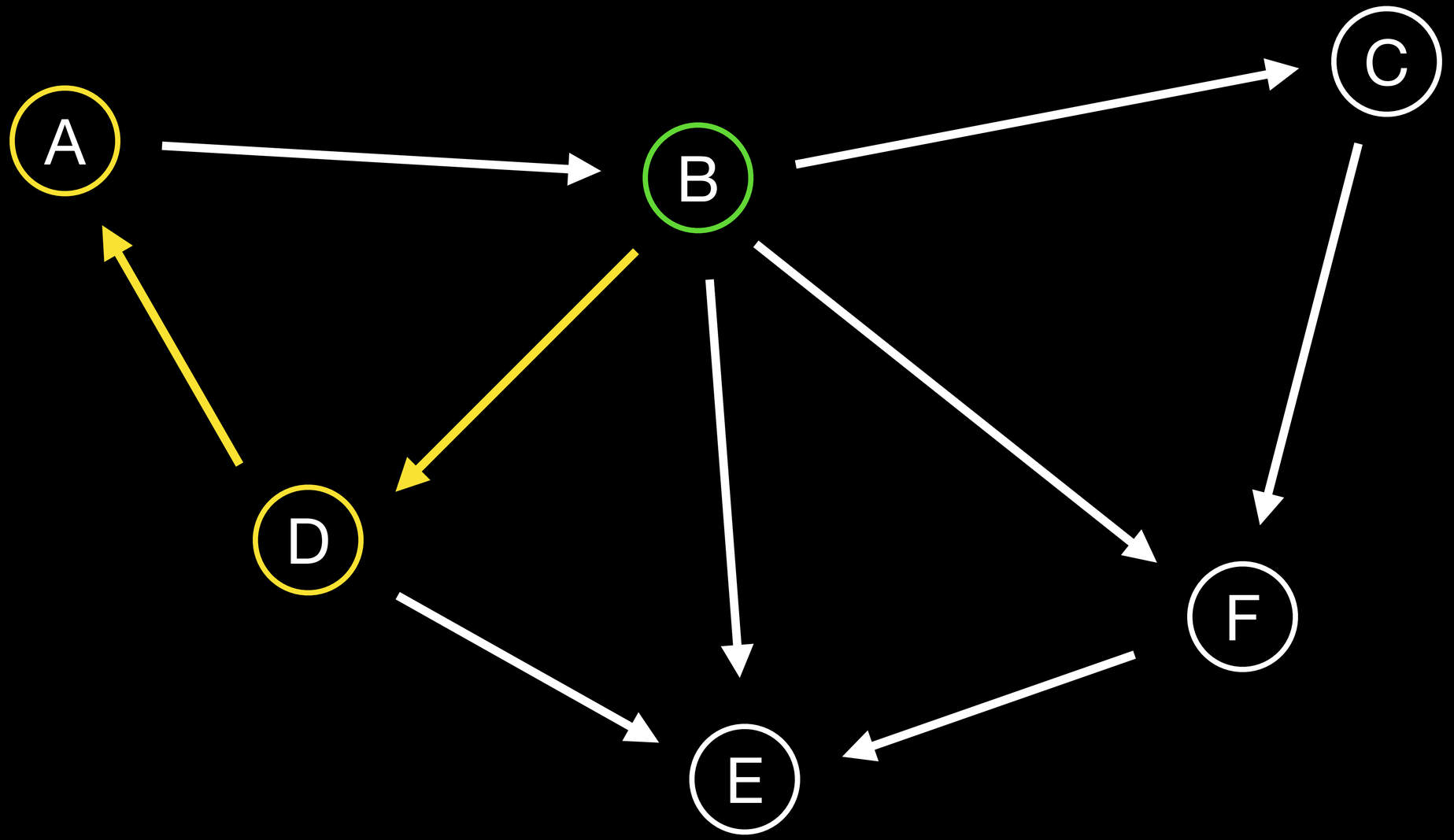
https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools

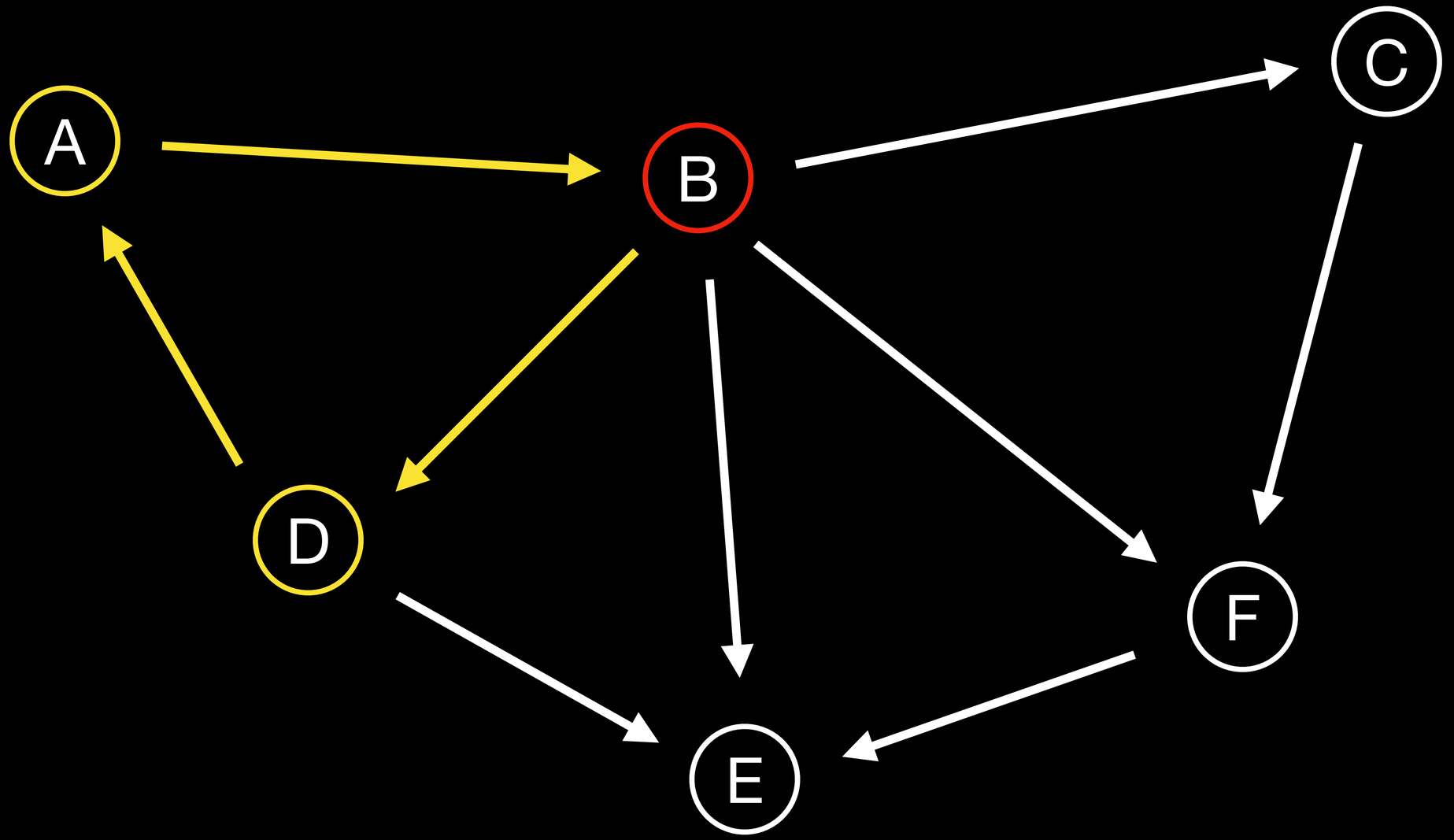
# Hamiltonian path

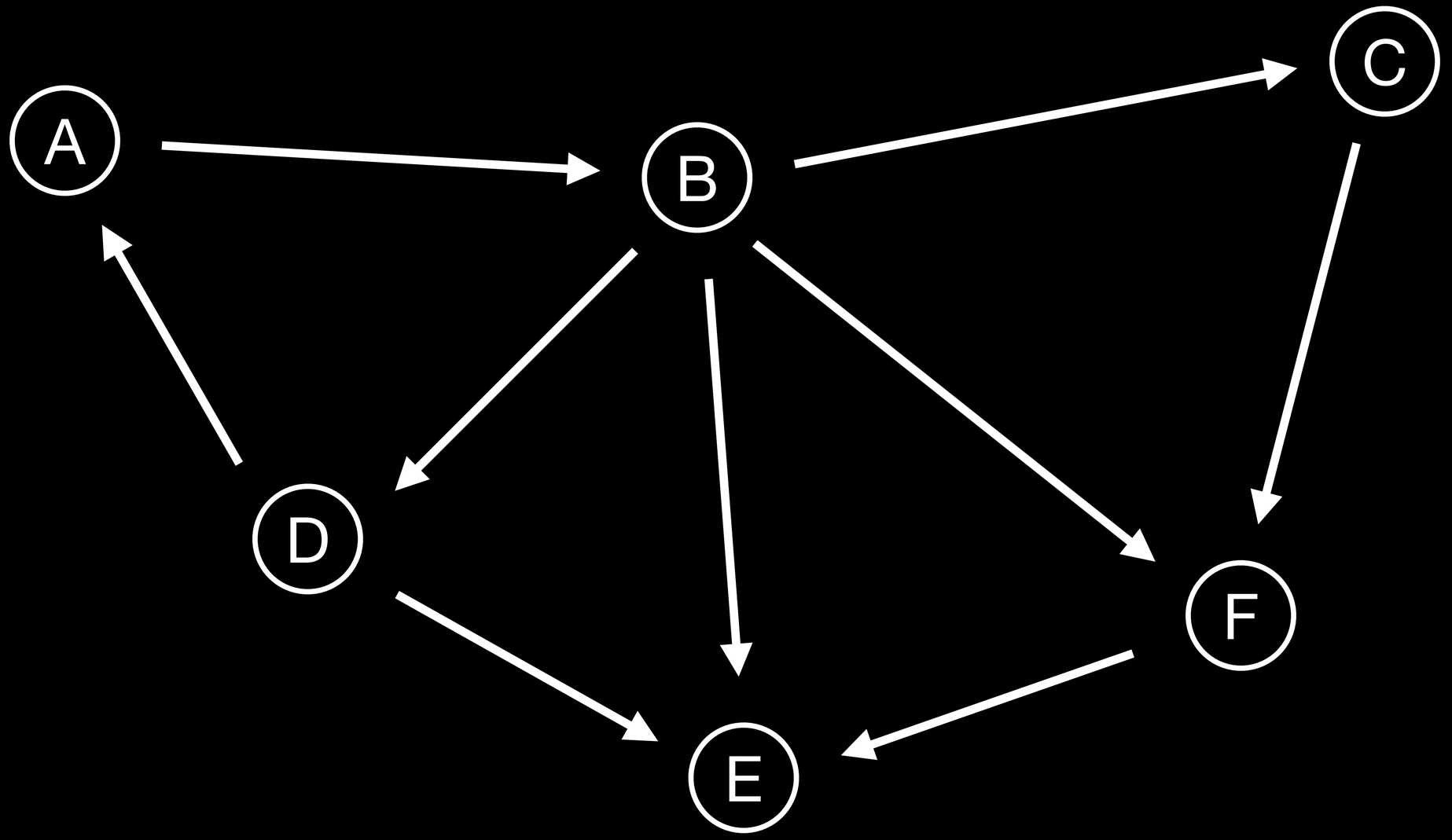

A
B
C
D
E
F

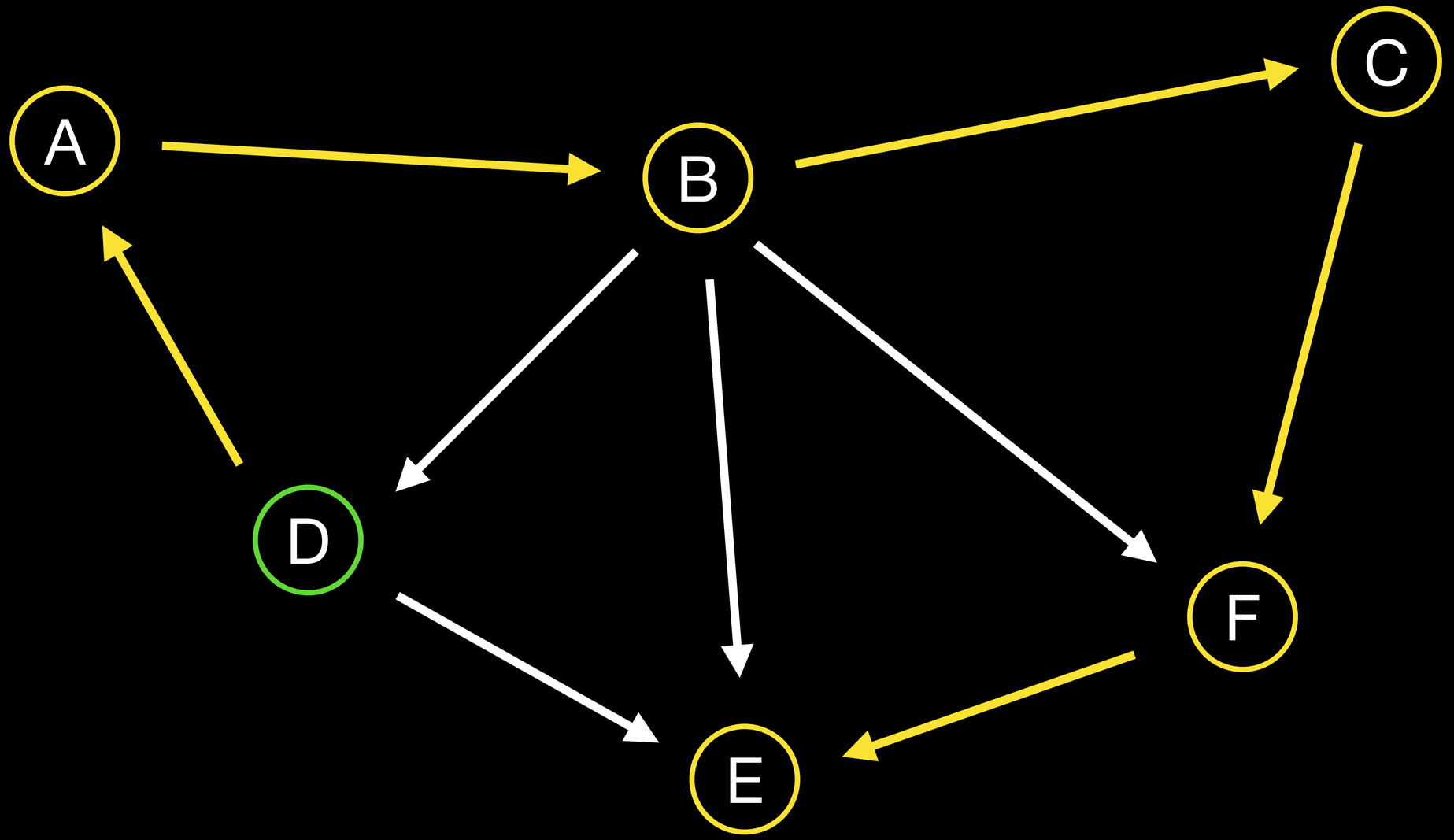

A
B
C
D
E
F

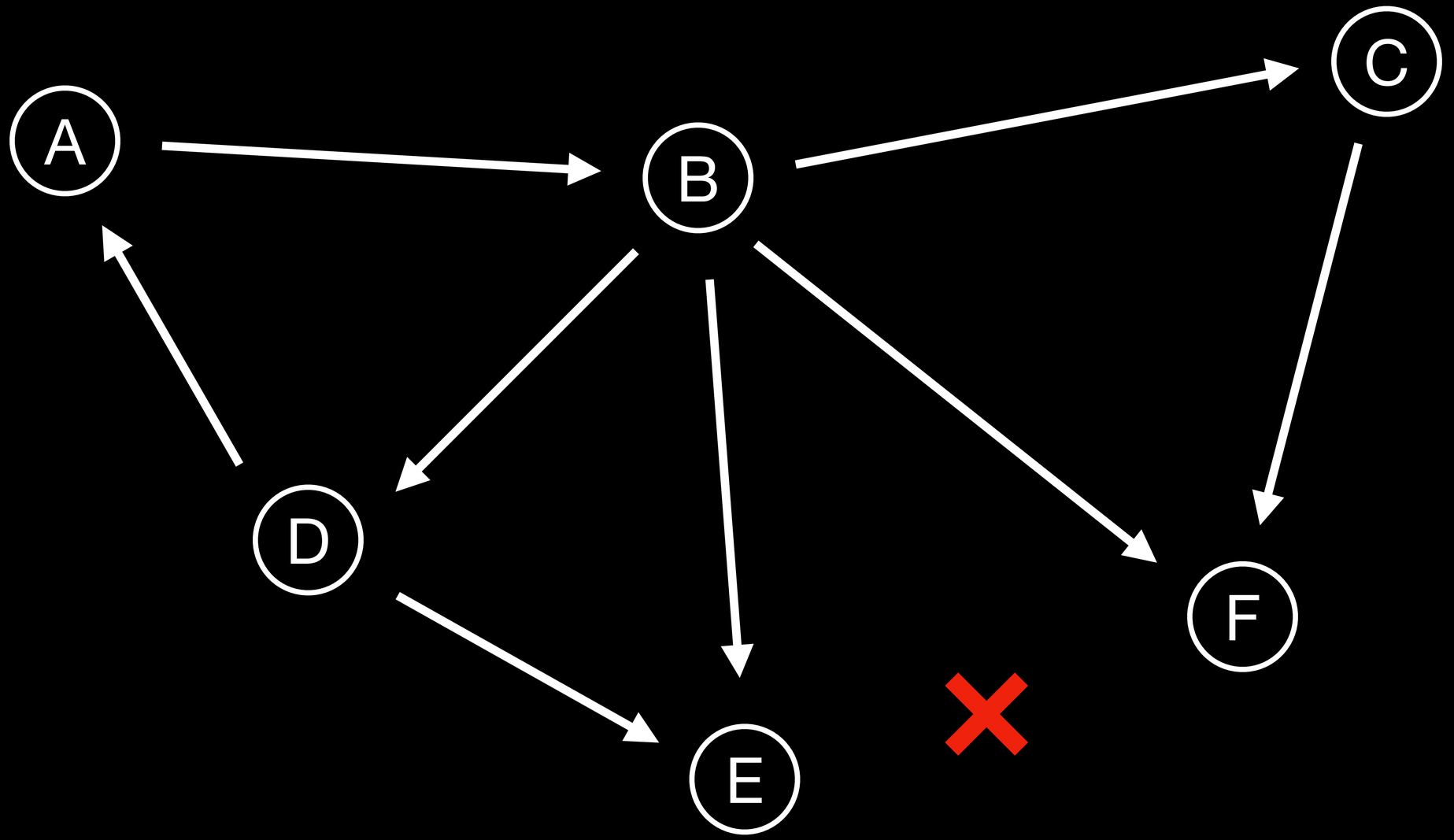

A
B
C
D
E
F


A
B
C
D
E
F


A
B
C
D
E
F


A
B
C
D
E
F


A
B
C
D
E
F


A
B
C
D
E
F

# Assembly

# How to signal if a path exists?

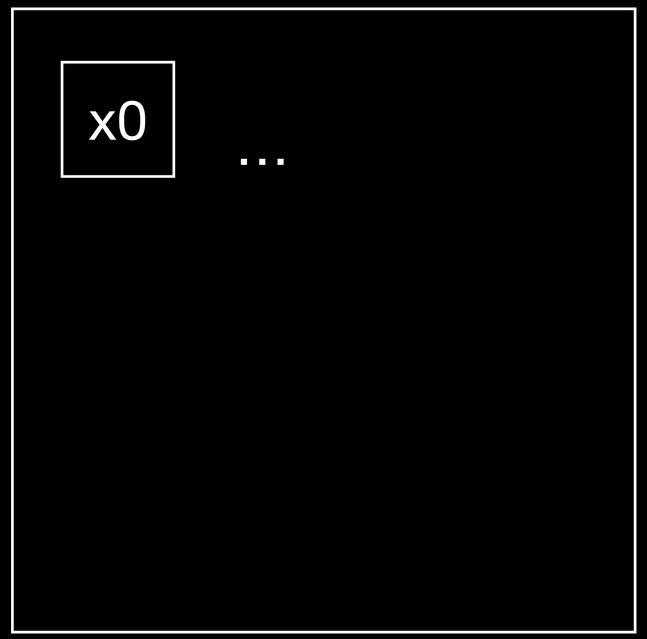
# How to signal if a path exists?

Exit status (0 or 1)

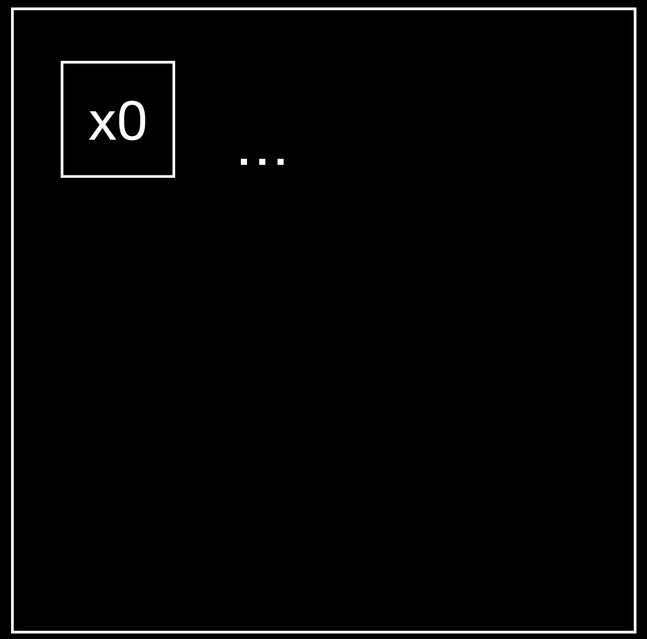
```
mov x16, #1
mov x0, #0
svc 0
```

# How to store the “visited” set?

# How to store the “visited” set?

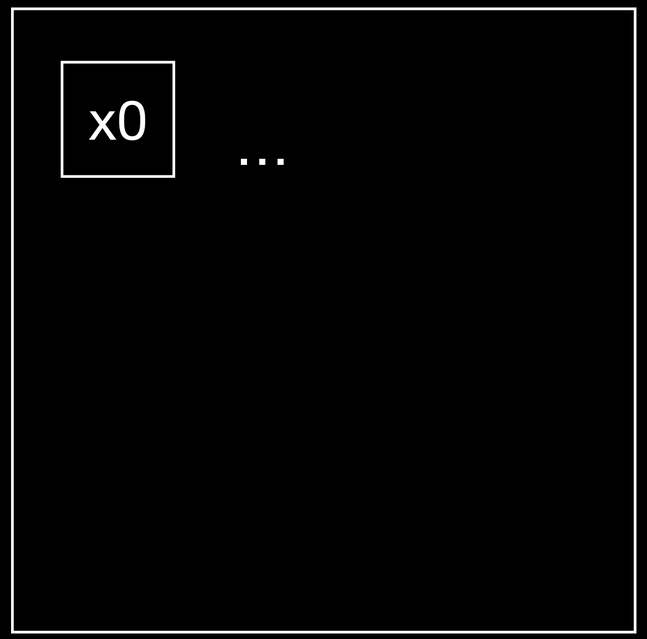


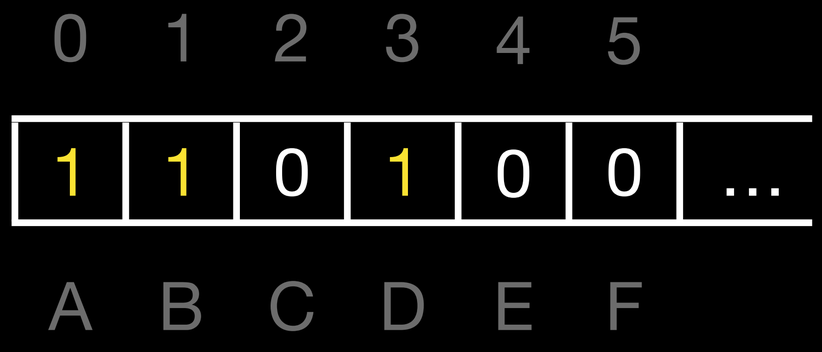
# How to store the “visited” set?




1 cycle

# How to store the “visited” set?

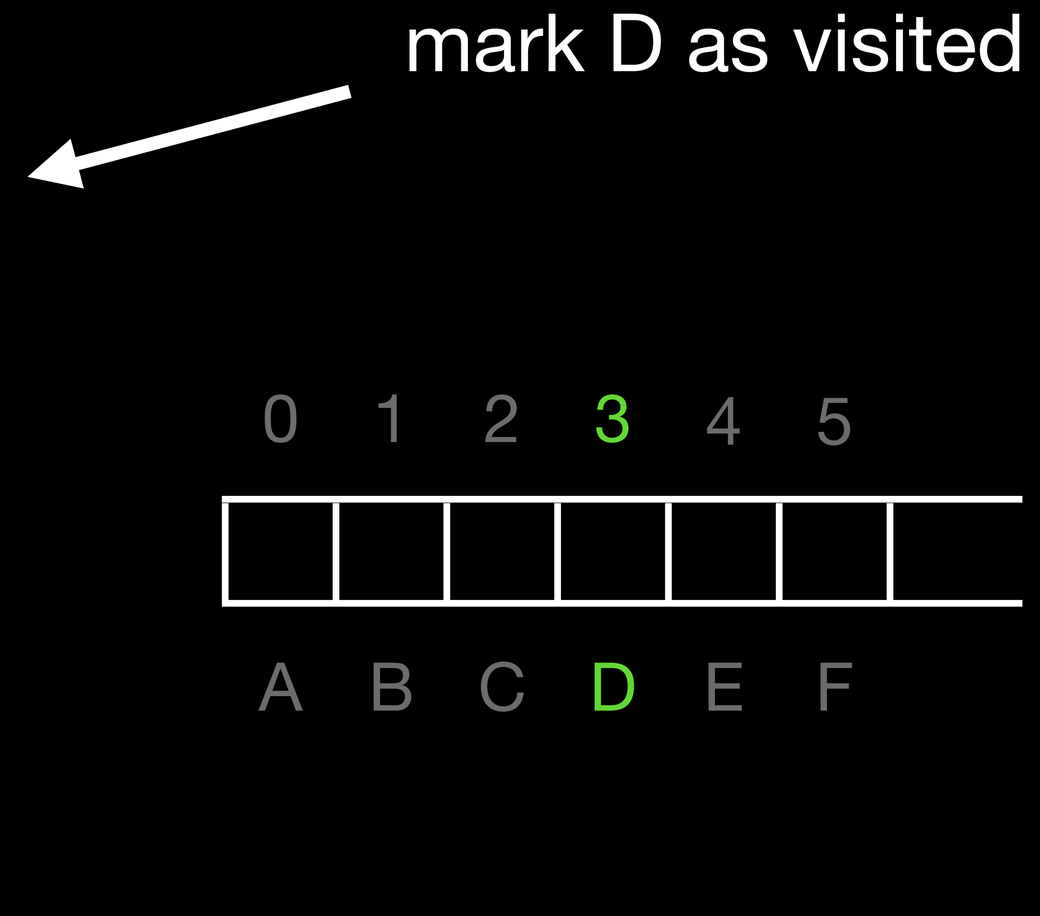


```
mov x0, #0
```

```
eor x0, x0, #8
```

```
eor x0, x0, #8
```



```
eor x0, x0, #8
```

mark D as unvisited

(flips the bit back again)

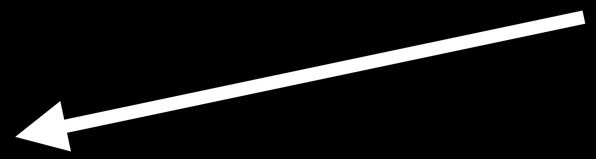
# How to visit a node?

# How to visit a node?

Branch to a subroutine

```
_main:
  bl visit_a

visit_a:
  // Visiting node A
```

```
_main:
  bl visit_a

visit_a:
  // Visiting node A
```



The “visit_a” subroutine

```
_main:
  bl visit_a

visit_a:
  // Visiting node A
```

# What is “branch link” ?

Moves the program counter

Sets the x30 register

```
_main:
  bl visit_a

visit_a:
  // Visiting node A
  ret
```

# How to branch conditionally?

Multiple different ways

```
tbz x0, #3, visit_d
```

```
tbz x0, #3, visit_d
```

test if a bit is zero

```
tbz x0, #3, visit_d
```

index 3

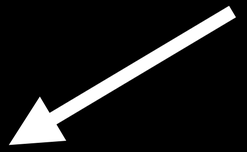
Problem: doesn’t set x30

```
tbnz x0, #3, after
bl visit_d
after:
```

test if non-zero

```
tbnz x0, #3, after
bl visit_d
after:
```

skip the next line



```
tbnz x0, #3, after
bl visit_d
after:
```

```
tbnz x0, #3, after
bl visit_d
after:
```

sets x30

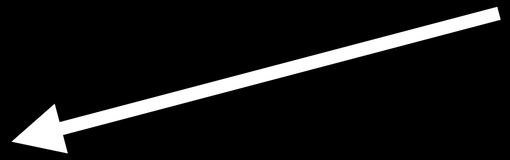
# How to check if we found a path?

The register is ‘111111’

```
cmp x0, #63
b.eq found_a_path
```


```
cmp x0, #63
b.eq found_a_path
```

```
cmp x0, #63
b.eq found_a_path
```



# Putting it together

```
_main:
  mov x0, #0

  bl visit_a
  bl visit_b
  bl visit_c
  bl visit_d
  bl visit_e
  bl visit_f

no_path_found:
  mov x16, #1
  mov x0, #1
  svc 0

found_a_path:
  mov x16, #1
  mov x0, #0
  svc 0
```

```
_main:
  mov x0, #0

  bl visit_a
  bl visit_b
  bl visit_c
  bl visit_d
  bl visit_e
  bl visit_f

no_path_found:
  mov x16, #1
  mov x0, #1
  svc 0

found_a_path:
  mov x16, #1
  mov x0, #0
  svc 0
```

We haven’t visited any nodes yet

```
_main:
  mov x0, #0

  bl visit_a
  bl visit_b
  bl visit_c
  bl visit_d
  bl visit_e
  bl visit_f

no_path_found:
  mov x16, #1
  mov x0, #1
  svc 0

found_a_path:
  mov x16, #1
  mov x0, #0
  svc 0
```

Try from every possible node

(do a depth-first search)

```
_main:
  mov x0, #0

  bl visit_a
  bl visit_b
  bl visit_c
  bl visit_d
  bl visit_e
  bl visit_f

no_path_found:
  mov x16, #1
  mov x0, #1
  svc 0

found_a_path:
  mov x16, #1
  mov x0, #0
  svc 0
```

Exit 1 if no path was found

```
_main:
  mov x0, #0

  bl visit_a
  bl visit_b
  bl visit_c
  bl visit_d
  bl visit_e
  bl visit_f

no_path_found:
  mov x16, #1
  mov x0, #1
  svc 0

found_a_path:
  mov x16, #1
  mov x0, #0
  svc 0
```

Branch here if we find a path

visit_a

```
visit_a:
  eor x0, x0, #1

  tbnz x0, #1, after_ab
  bl visit_b
  after_ab:

  cmp x0, #63
  b.eq found_a_path

  eor x0, x0, #1
  ret
```

```
visit_a:
  eor x0, x0, #1
```

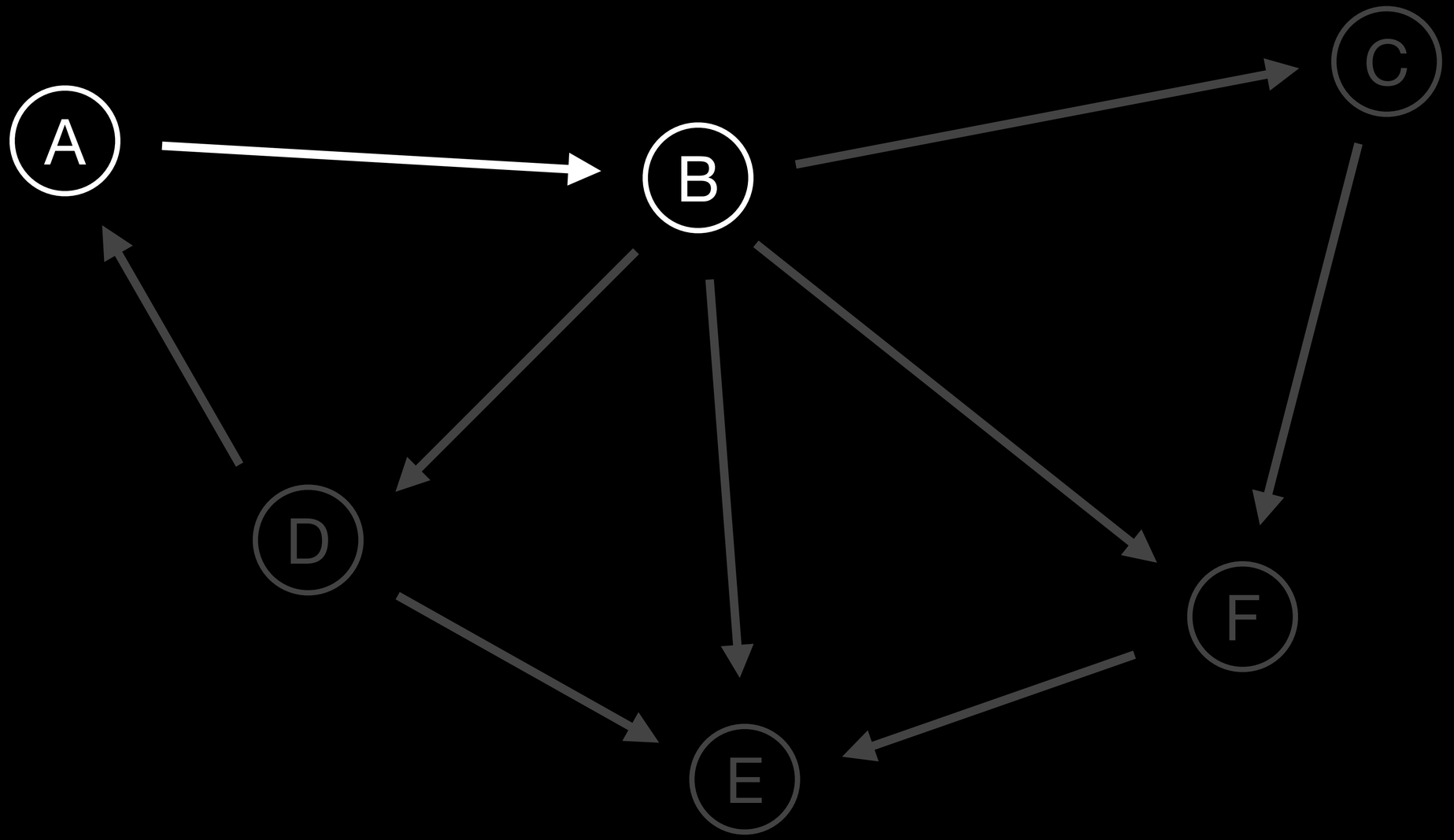
Mark A as visited: $2^0$

```
  tbnz x0, #1, after_ab
  bl visit_b
  after_ab:

  cmp x0, #63
  b.eq found_a_path

  eor x0, x0, #1
  ret
```

```
visit_a:
  eor x0, x0, #1

  tbnz x0, #1, after_ab
  bl visit_b
  after_ab:

  cmp x0, #63
  b.eq found_a_path

  eor x0, x0, #1
  ret
```

Skip b if already visited

(index 1)


A
B
C
D
E
F

```
visit_a:
  eor x0, x0, #1

  tbnz x0, #1, after_ab
  bl visit_b
  after_ab:

  cmp x0, #63
  b.eq found_a_path

  eor x0, x0, #1
  ret
```

Check if all nodes visited

```
visit_a:
  eor x0, x0, #1

  tbnz x0, #1, after_ab
  bl visit_b
  after_ab:

  cmp x0, #63
  b.eq found_a_path

  eor x0, x0, #1
  ret
```

Mark A as unvisited: $2^0$

Repeat for other nodes

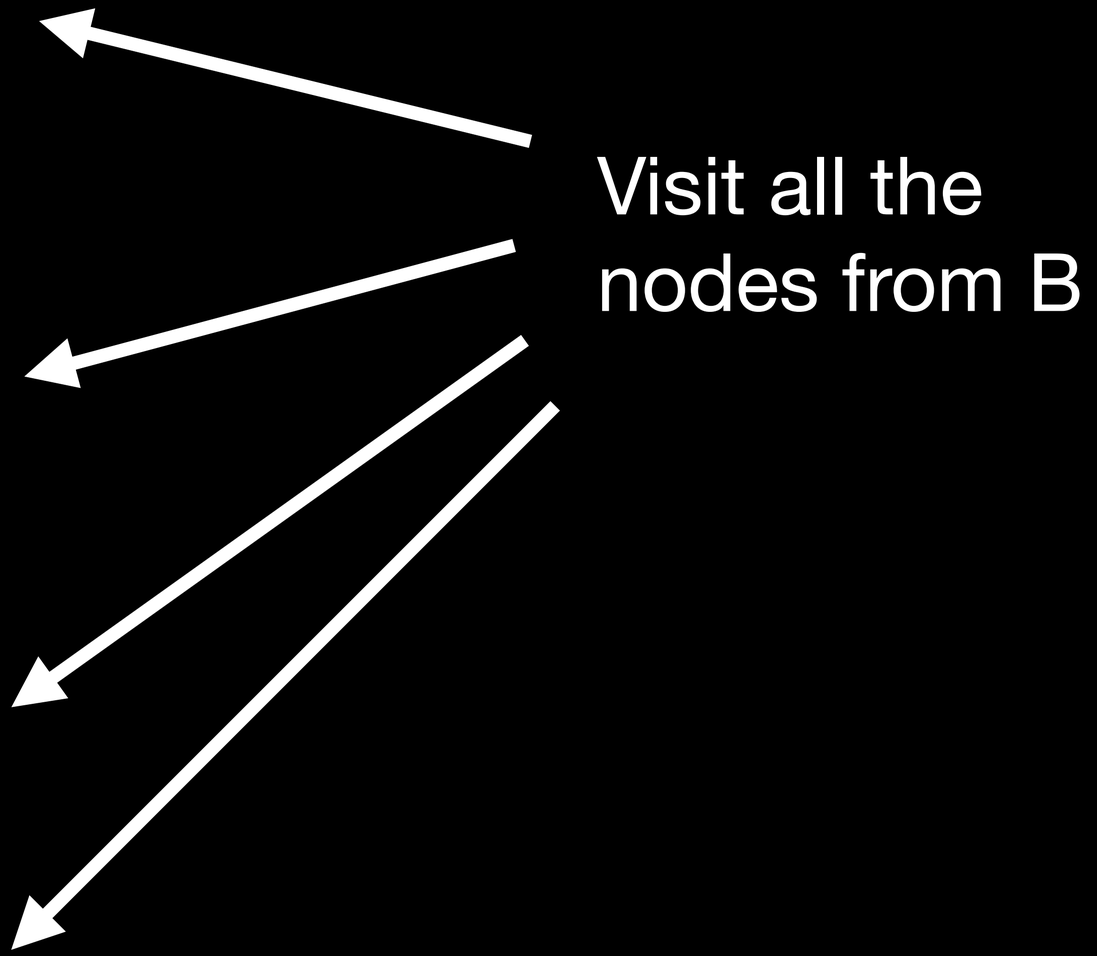
```
visit_b:
  eor x0, x0, #2

  tbnz x0, #2, after_bc
  bl visit_c
  after_bc:

  tbnz x0, #3, after_bd
  bl visit_d
  after_bd:

  // ...
```

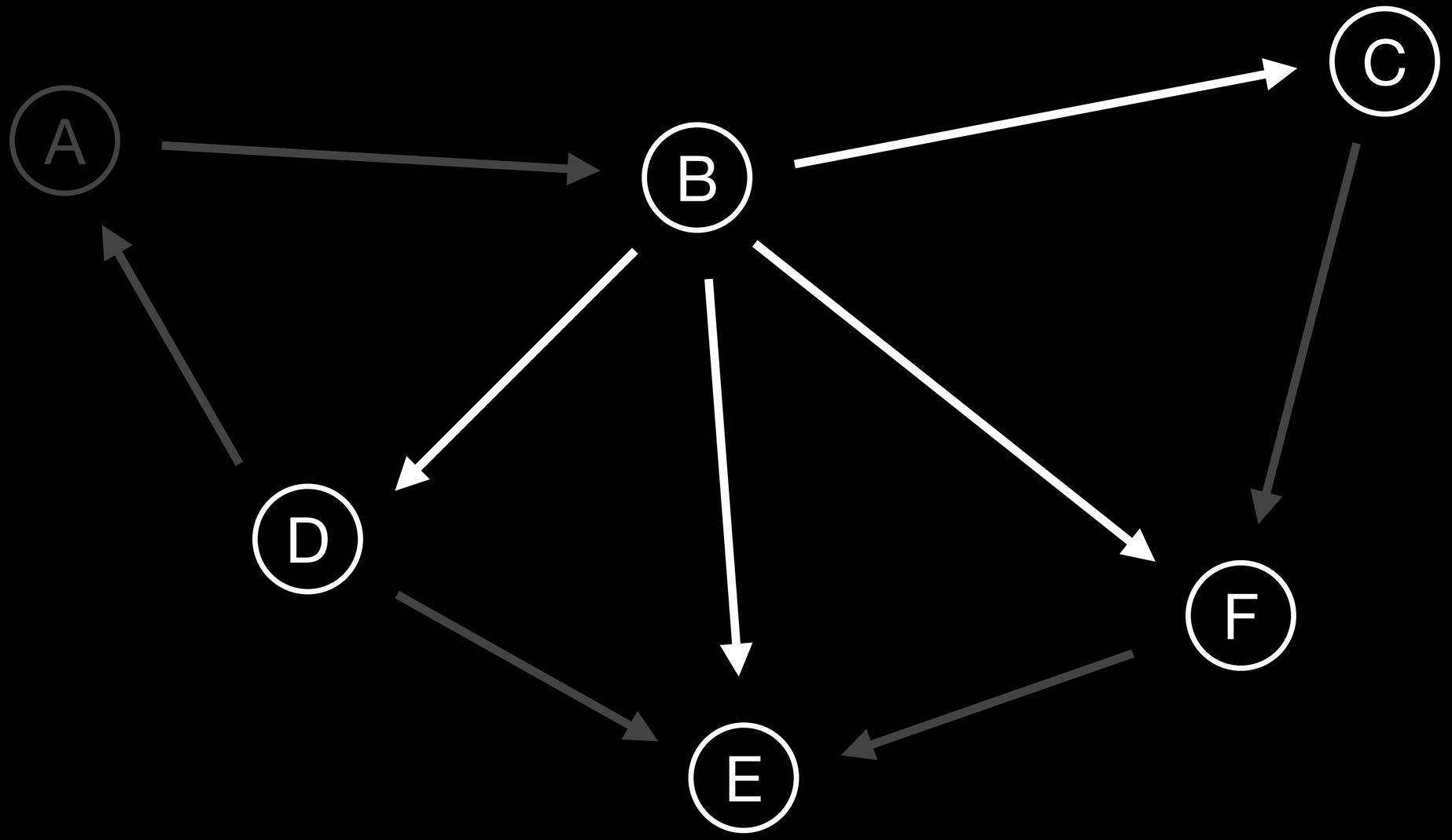
```
visit_b:
  eor x0, x0, #2
```

Mark B as visited: $2^1$

```
  tbnz x0, #2, after_bc
  bl visit_c
  after_bc:

  tbnz x0, #3, after_bd
  bl visit_d
  after_bd:

  // ...
```

```
visit_b:
  eor x0, x0, #2

  tbnz x0, #2, after_bc
  bl visit_c
  after_bc:

  tbnz x0, #3, after_bd
  bl visit_d
  after_bd:

  // ...
```




A
B
C
D
E
F

# And that’s it!

And that’s it!

Well, not quite!

And that’s it!

Well, not quite!

There’s a problem

```
visit_a
    visit_b
        visit_d
```

visit_a

visit_b

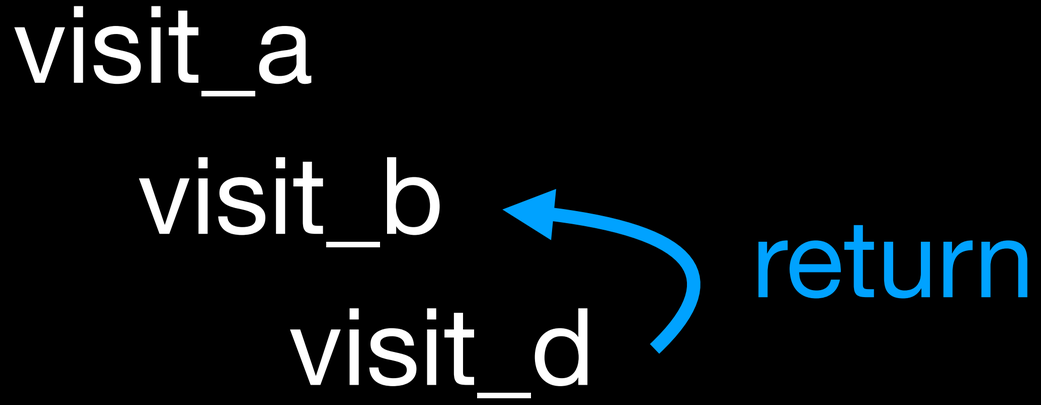
visit_d

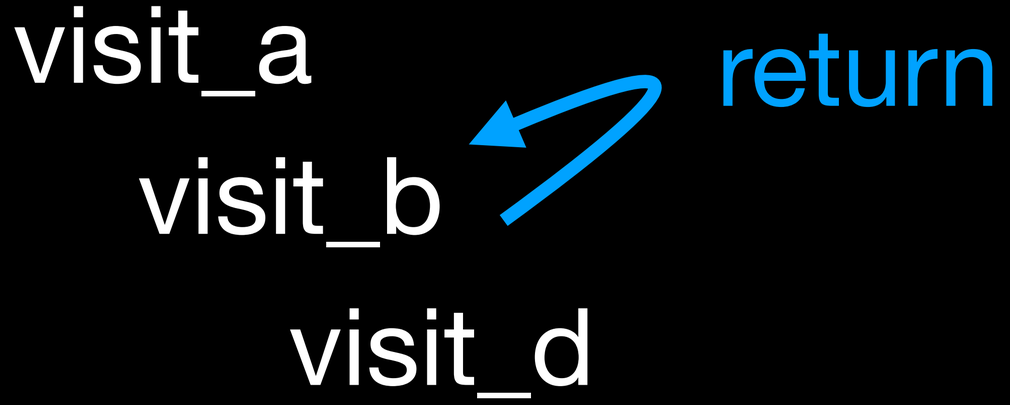
call stack

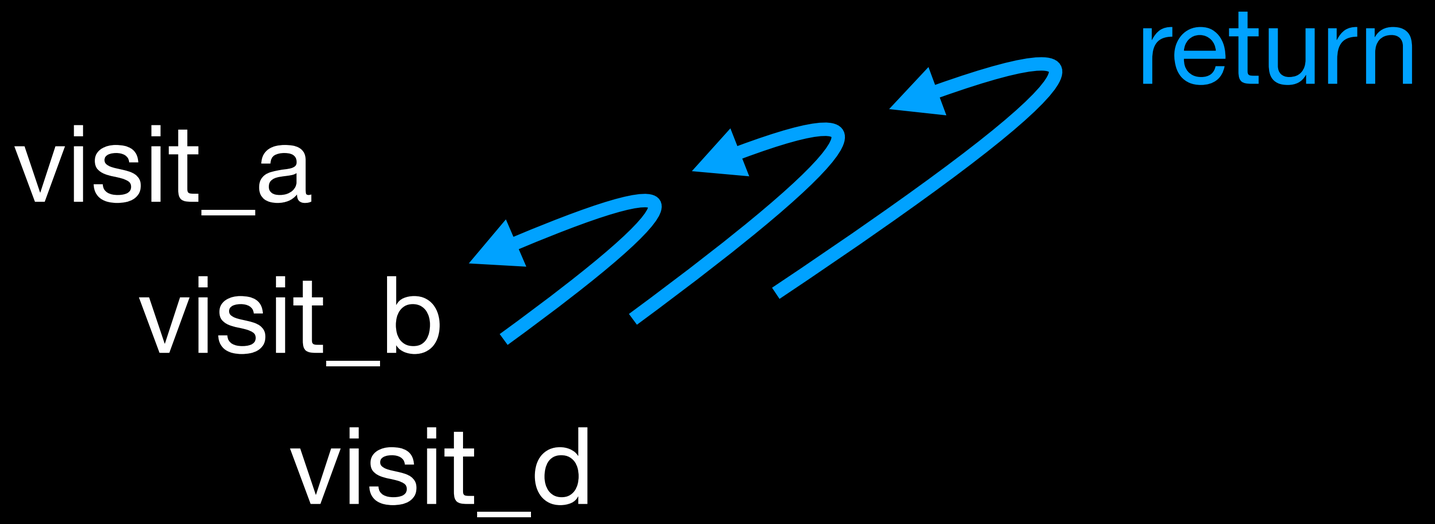
visit_a sets x30

visit_b sets x30

visit_d


visit_a
visit_b
visit_d
return


visit_a
visit_b
visit_d
return

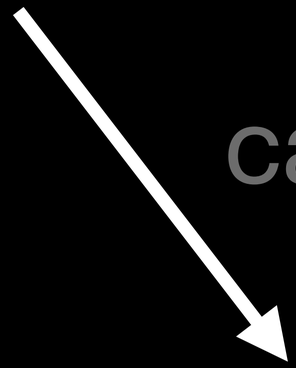

return
visit_a
visit_b
visit_d

We need a stack!

visit_a

visit_b

visit_d



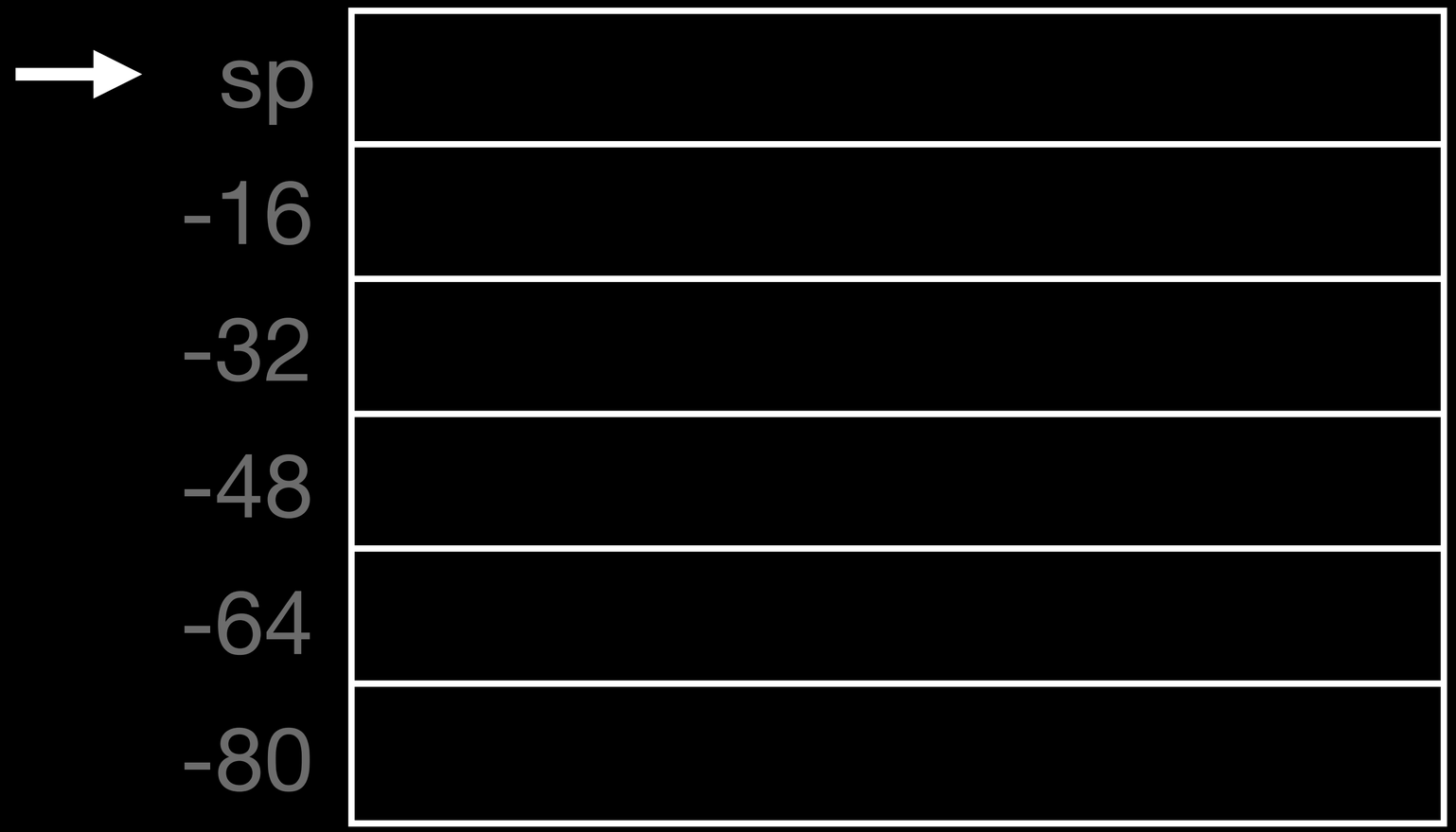
call stack

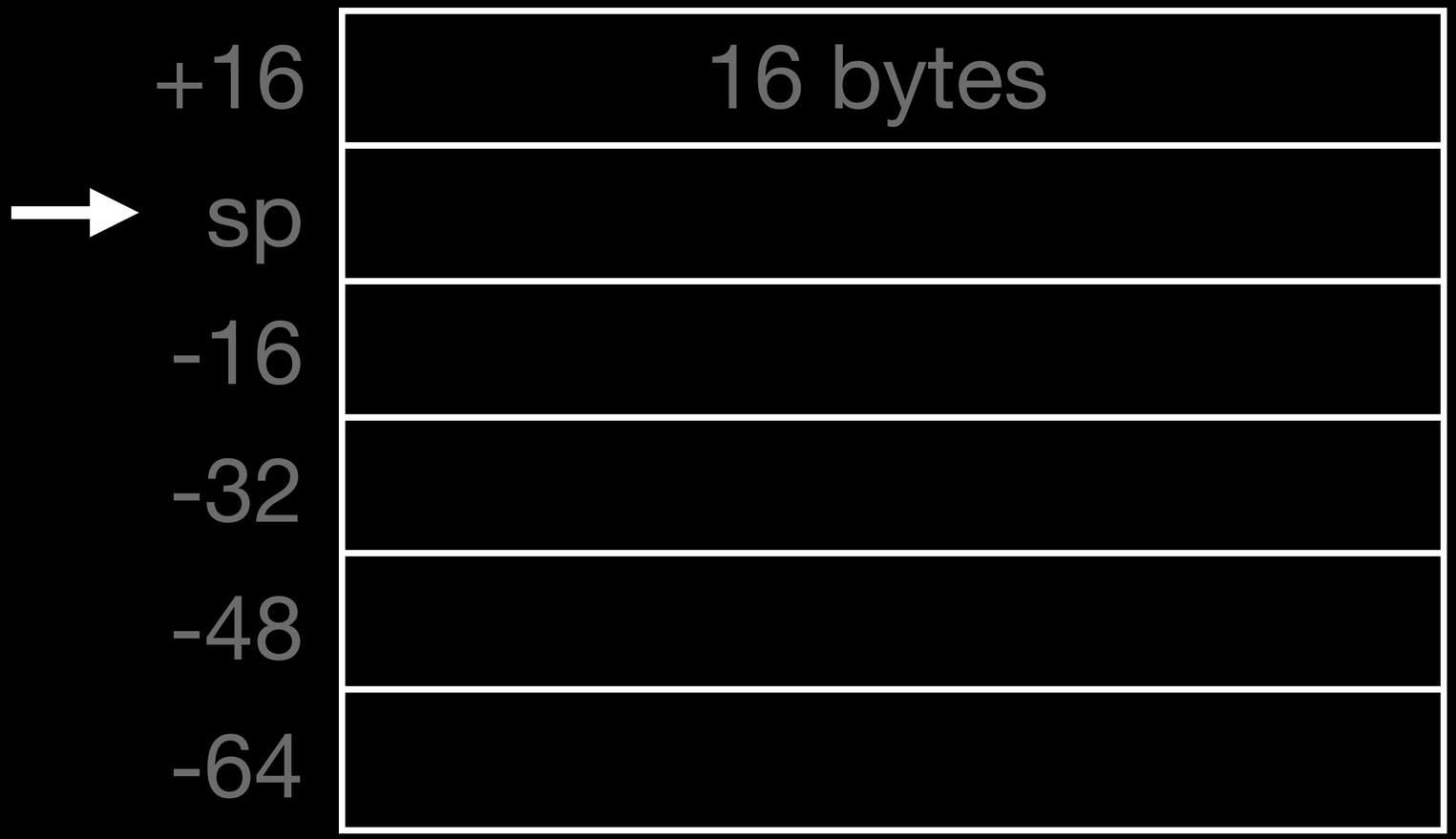
# How the stack works

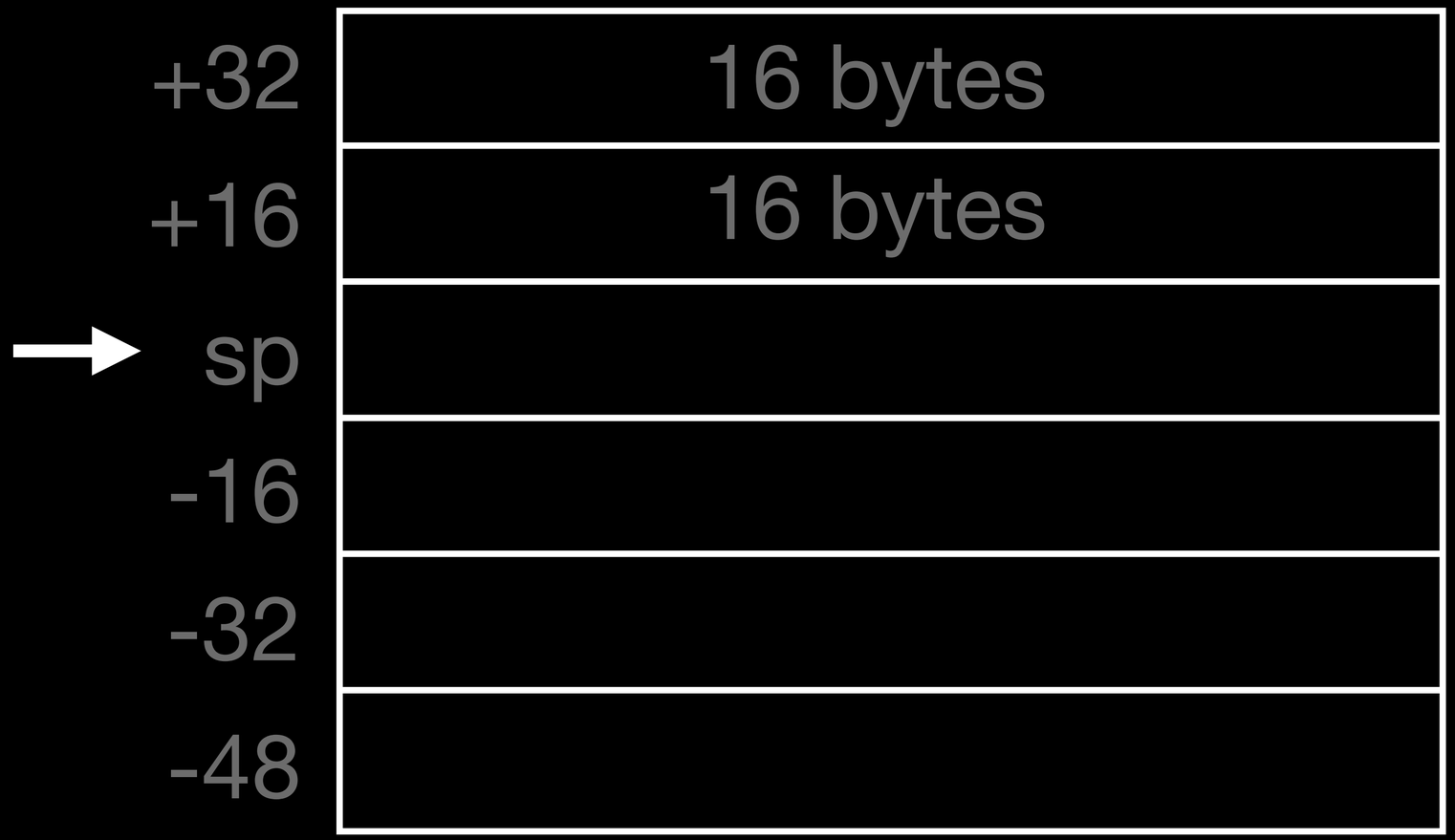
Stored in memory

Stack pointer (sp) register

Address moves down


sp
-16
-32
-48
-64
-80


+16
16 bytes
sp
-16
-32
-48
-64


+32
16 bytes
+16
16 bytes
sp
-16
-32
-48

```
str x30, [sp, -16]!
ldr x30, [sp], 16
```

```
str x30, [sp, -16]!
ldr x30, [sp], 16
```

```
str x30, [sp, -16]!
ldr x30, [sp], 16
```

```
visit_a:
  str x30, [sp, -16]!

  // ...

  ldr x30, [sp], 16
  ret
```

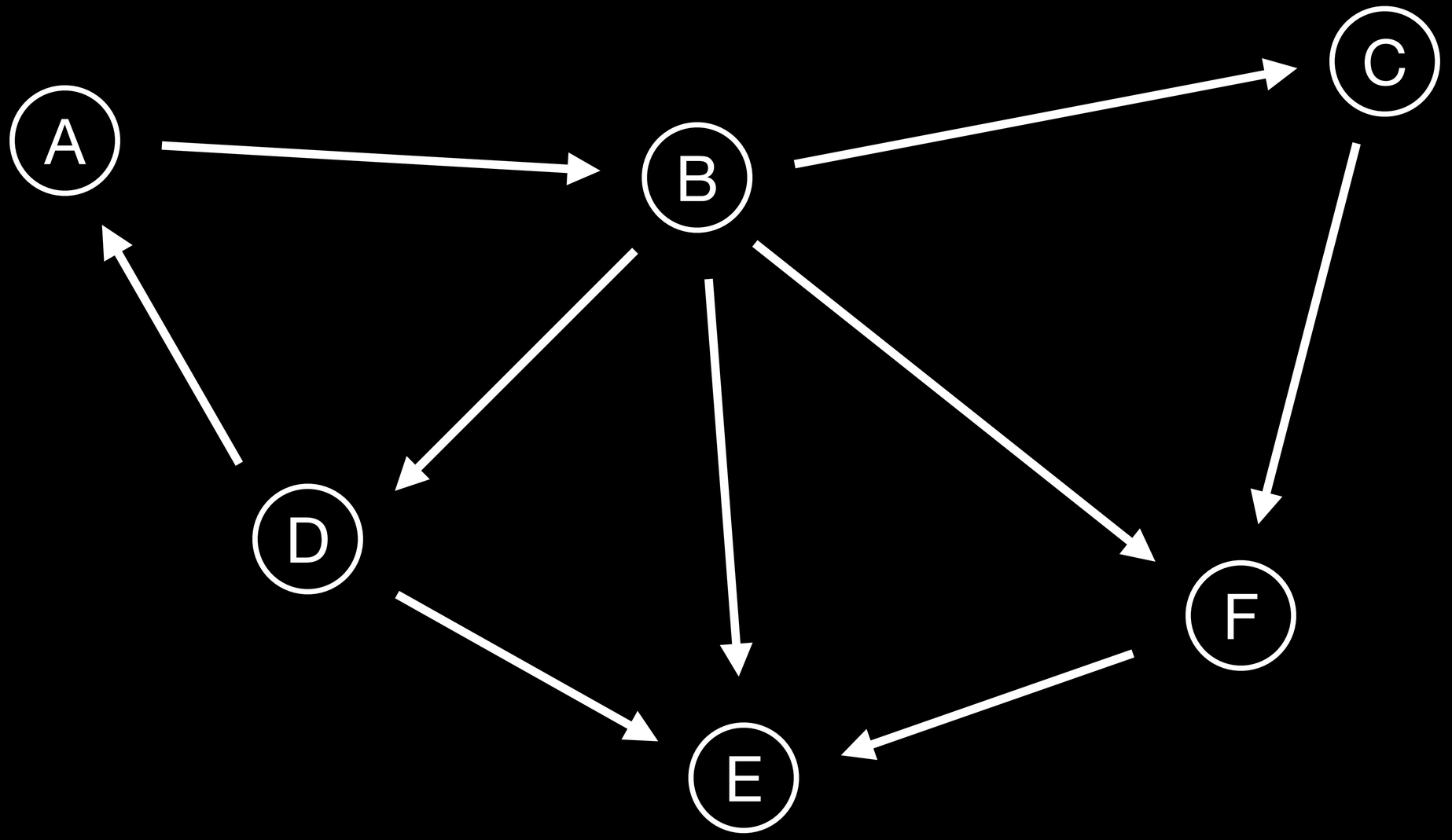
Code from before

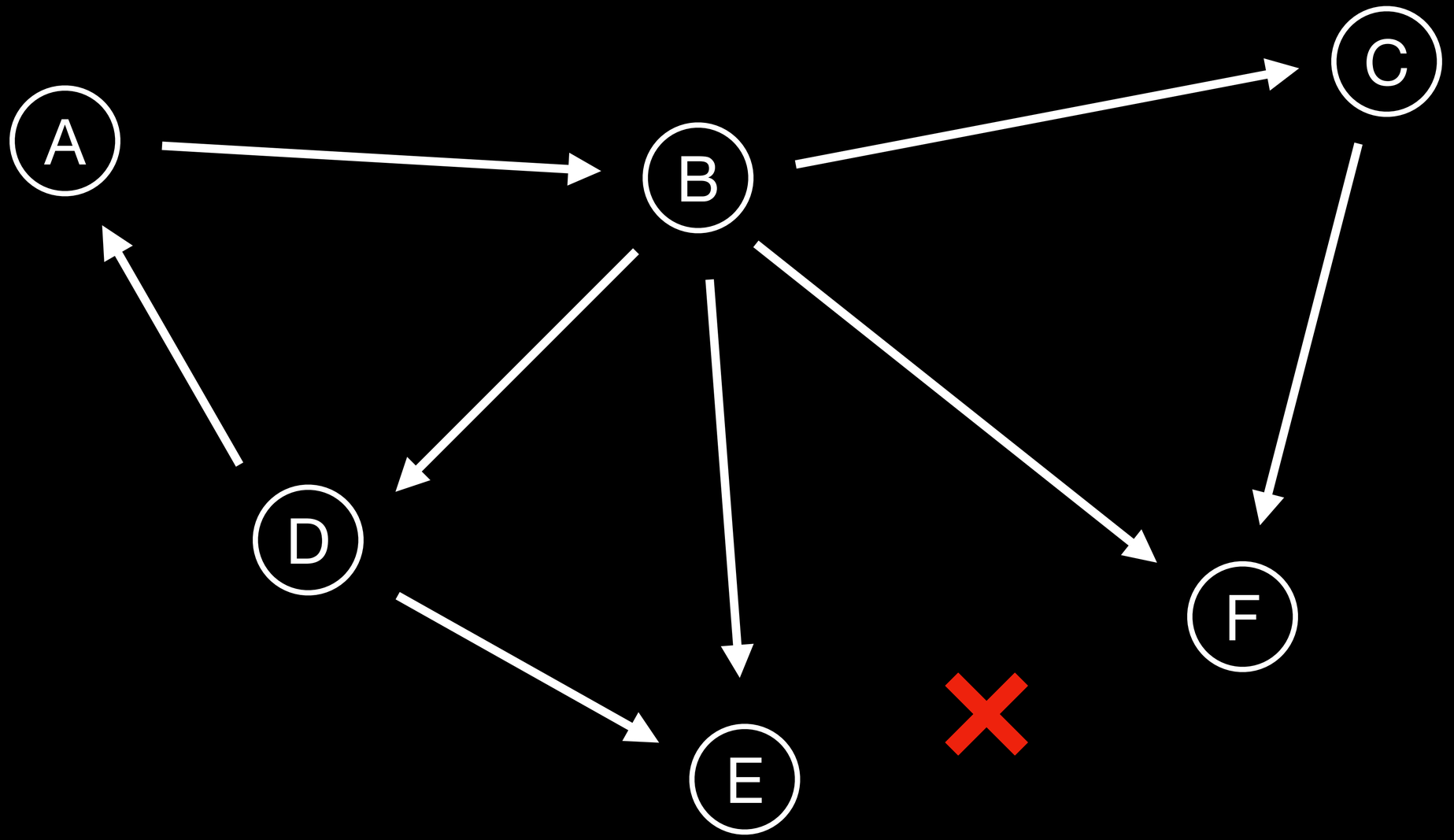
# And that’s it!

(really)

# Demo

https://github.com/tuzz/assembly/blob/main/src/hamiltonian.s


A
B
C
D
E
F


A
B
C
D
E
F

# By the end

Access to a powerful tool

Know more about your computer

Where to go next

# Useful resources

https://modexp.wordpress.com/2018/10/30/arm64-assembly

https://www.youtube.com/watch?v=GBRdzaAxHB8

https://www.amazon.co.uk/dp/1484258800

https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools

# Thanks!

@chrispatuzzo