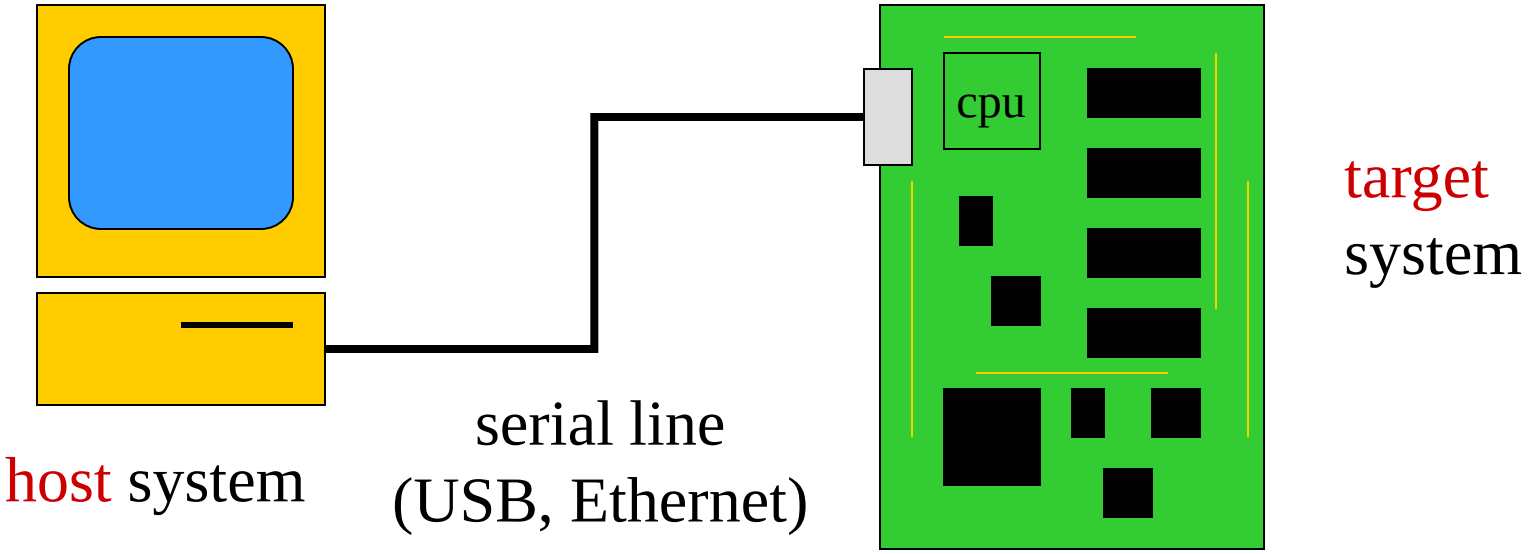
4.6 Debugging embedded systems

- Challenges:
  - target system may be hard to observe;
  - target may be hard to control;
  - may be hard to generate realistic inputs;
  - setup sequence may be complex.

Host/target design

- Use a host system to prepare software for target system:



Host-based tools

- Cross compiler:
  - compiles code on host for target system.
- Cross debugger:
  - displays target state, allows target system to be controlled.
  - by establishing a debug message protocol and using an interface like TCP/IP for communication between host development system and the target system, where the application to be debugged actually runs.

Software for debuggers

- A monitor, which is a small debug handler application, should run in user space on the target. It usually idles in user space memory and gets triggered by a dedicated debug interrupt.
- This is when it starts sending status information via a dedicated TCP/IP port to the host system where the debugger itself is waiting to pick up the data it receives.

Software for debuggers

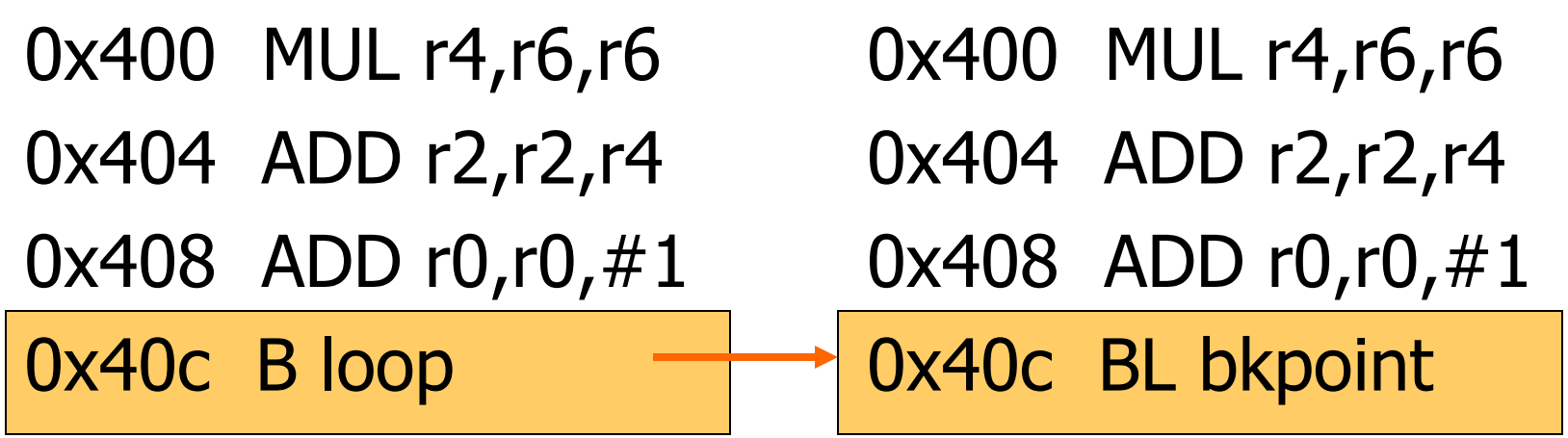
- The debug interrupt could be caused by a breakpoint or data watchpoint being hit. It could also be triggered by an explicit action on the debug host.
- The developer telling the debugger to attach to a specific running process or telling the debugger to stop a specific thread.

Breakpoints

- A breakpoint allows the user to stop execution, examine system state, and change state.
- Replace the breakpointed instruction with a subroutine call to the monitor program.
- Can you set breakpoints in programs running out of ROM?

  No, but ROM emulator can be used

ARM breakpoints



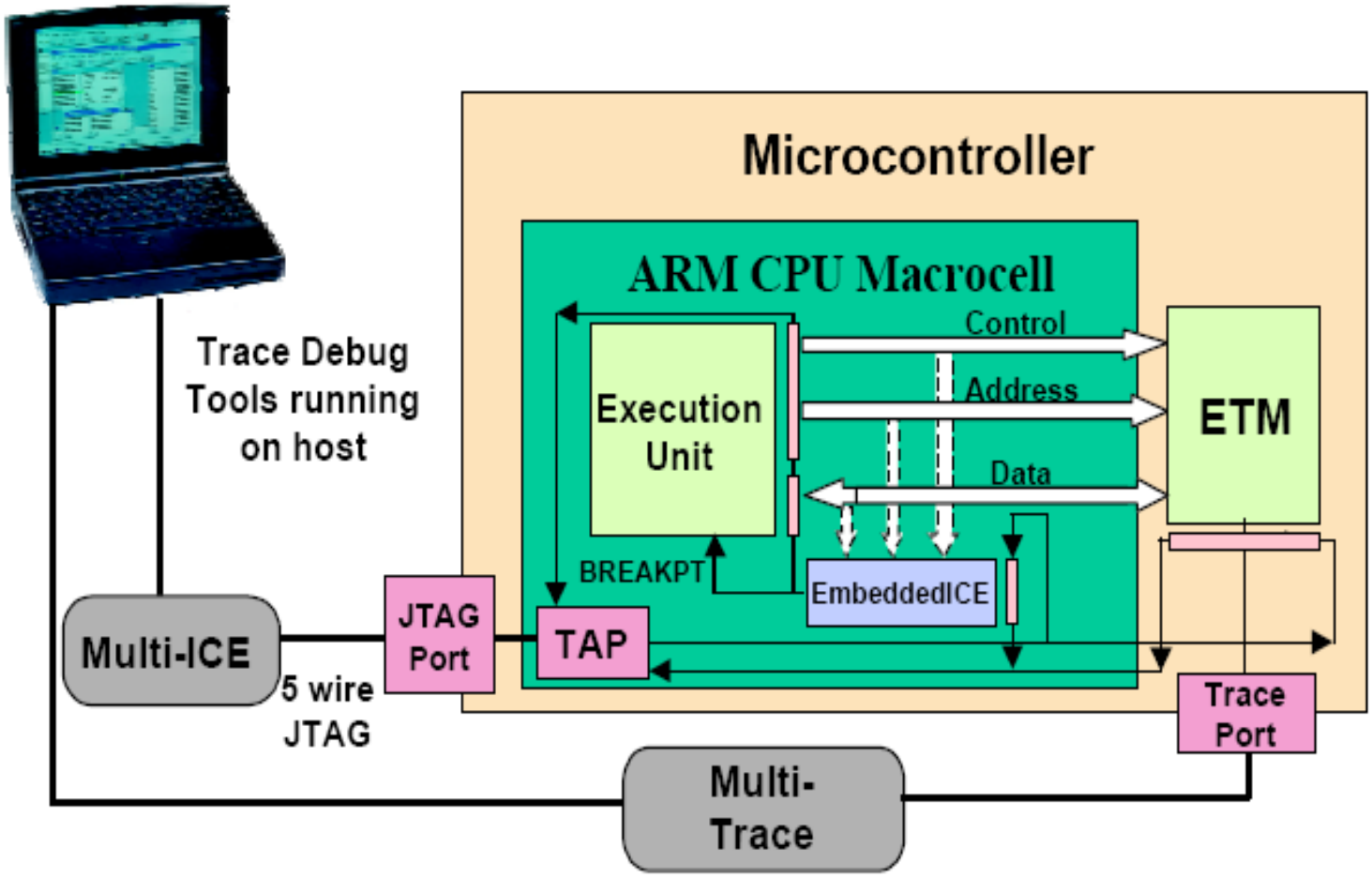
Breakpoint handler actions

- Save registers.
- Allow user to examine machine.
- Before returning, restore system state.
  - (when the breakpoint is erased) Safest way to continue execution is to replace back the original instruction while fixing the return address.
  - (when the breakpoint is to remain) Put another temp breakpoint after replacing back the original instruction. When reached to the temp breakpoint after executing the original instruction, replace back the original breakpoint, remove the temp breakpoint, and resume execution.

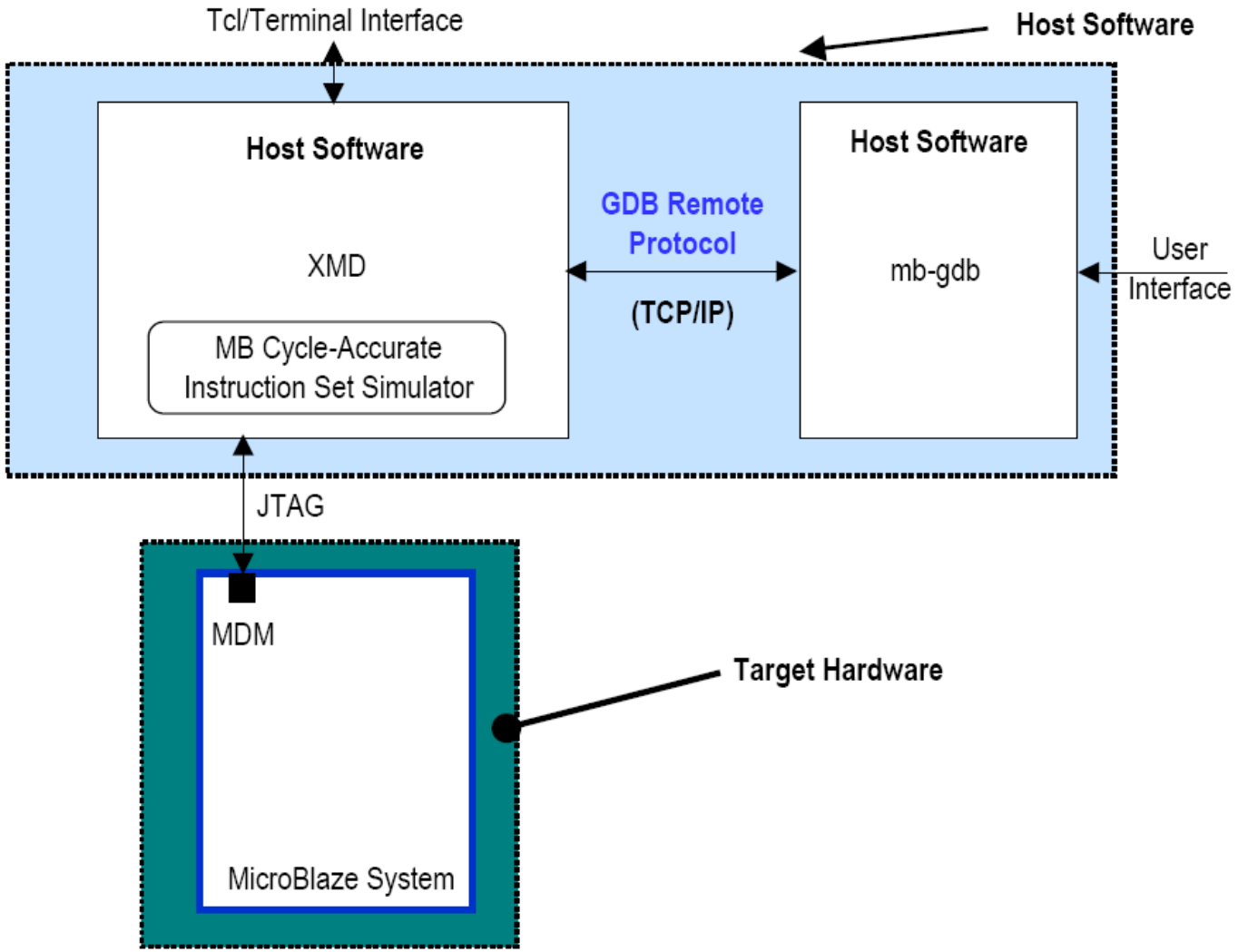
In-circuit emulators (ICE)

- A microprocessor in-circuit emulator is a specialized hardware tool to help debug in aspecially-instrumented microprocessor.
- Allows you to stop execution, examine CPU state, modify registers.
- the emulator is a bridge between your target and your PC, giving you both an interactive terminal peering deeply into the target, while providing a rich set of debugging resources.

Embedded ICE in ARM



Hardware or ISS target

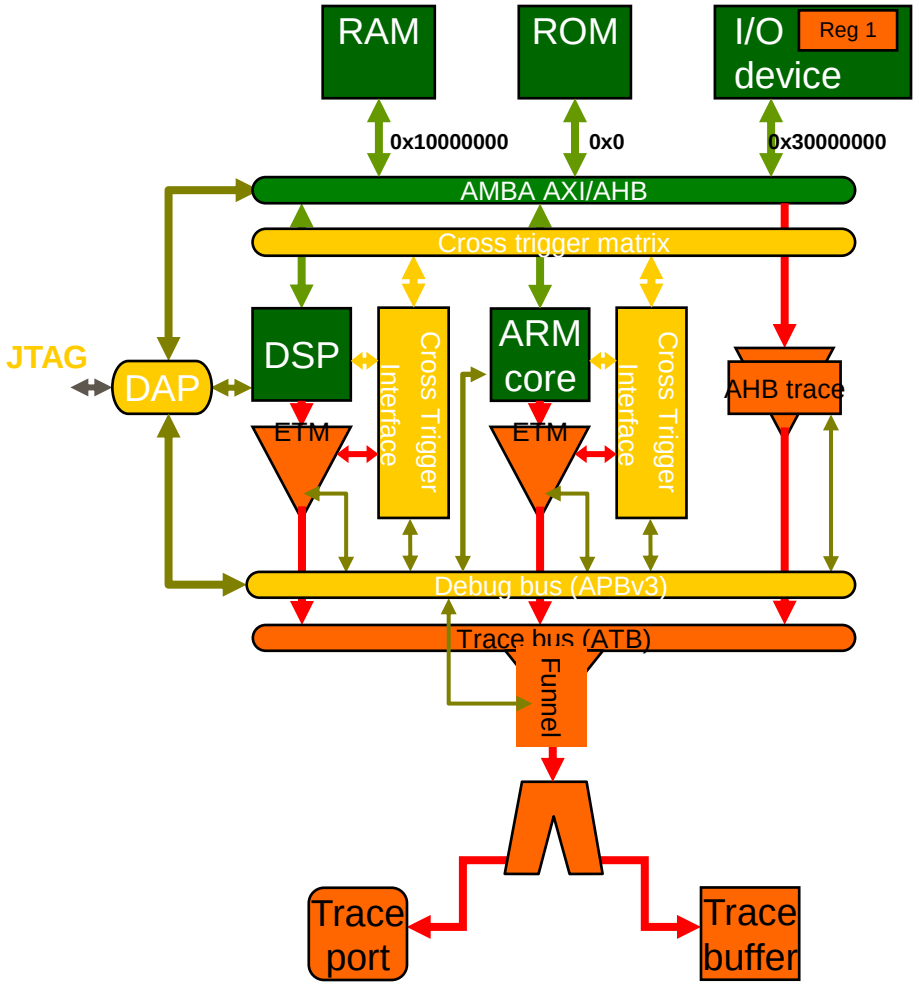


History

- In the beginning, there was the ROM debug monitor.
- After that the in-circuit emulator (ICE) came. By using special bond-out versions of processors, an ICE provides capabilities far beyond those of a simple ROM monitor.
- Now, dedicated debug circuitry is integrated into their chips. Or, simply software debug capabilities are added to their existing JTAG ports. Collectively, we'll call these technologies on-chip debug. Such hardware-based capabilities take the place of a software debug monitor, yet offer some additional features previously associated only with emulators.

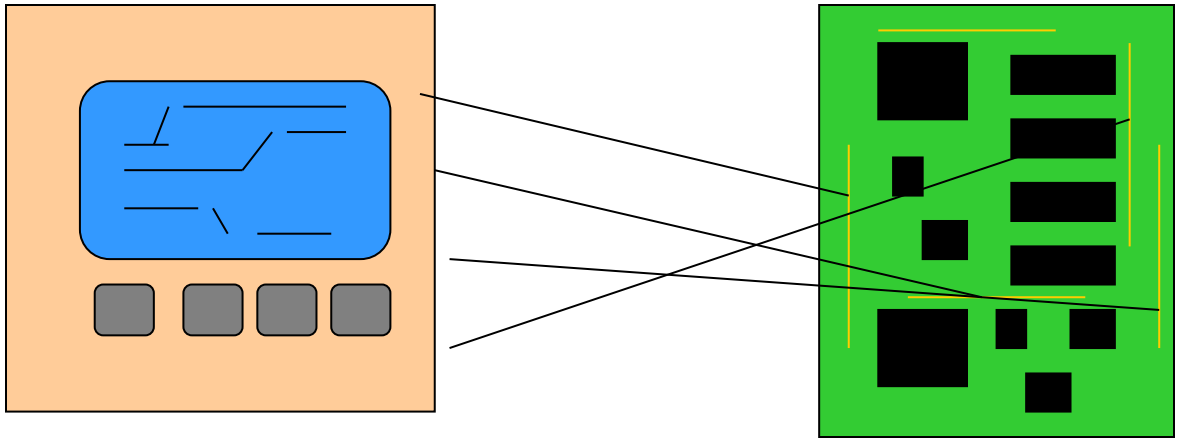
What does the debugger need to know?

- Programmers' model:
  - System components
  - System busses
  - Base addresses
  - Device registers
- Debug access description:
  - Debug access to processors
  - Other debug devices
  - Debug interconnections

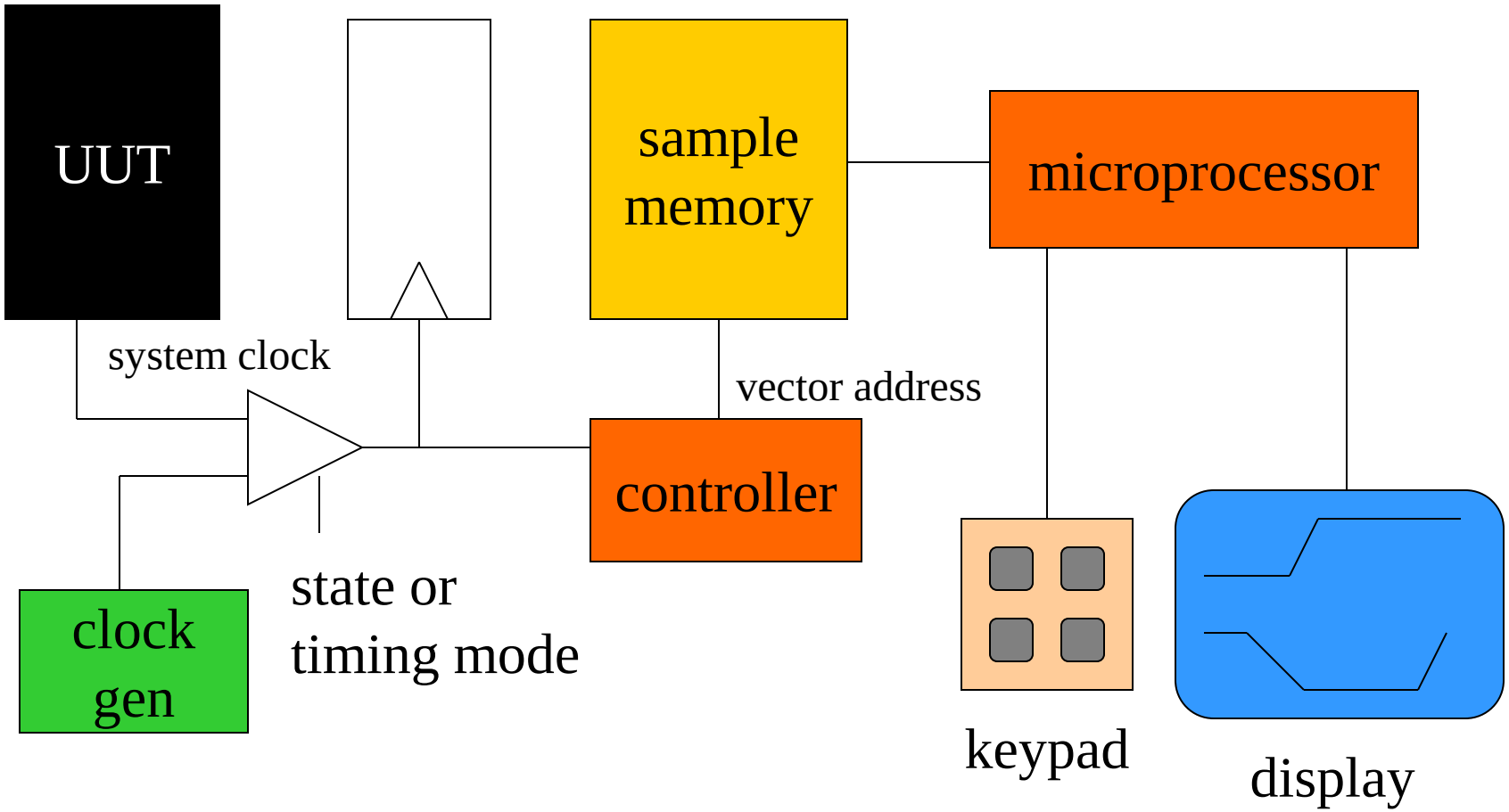


Logic analyzer

- A logic analyzer records the values of multiple channels into an internal memory and then display them on the display
  - State mode
  - Timing mode

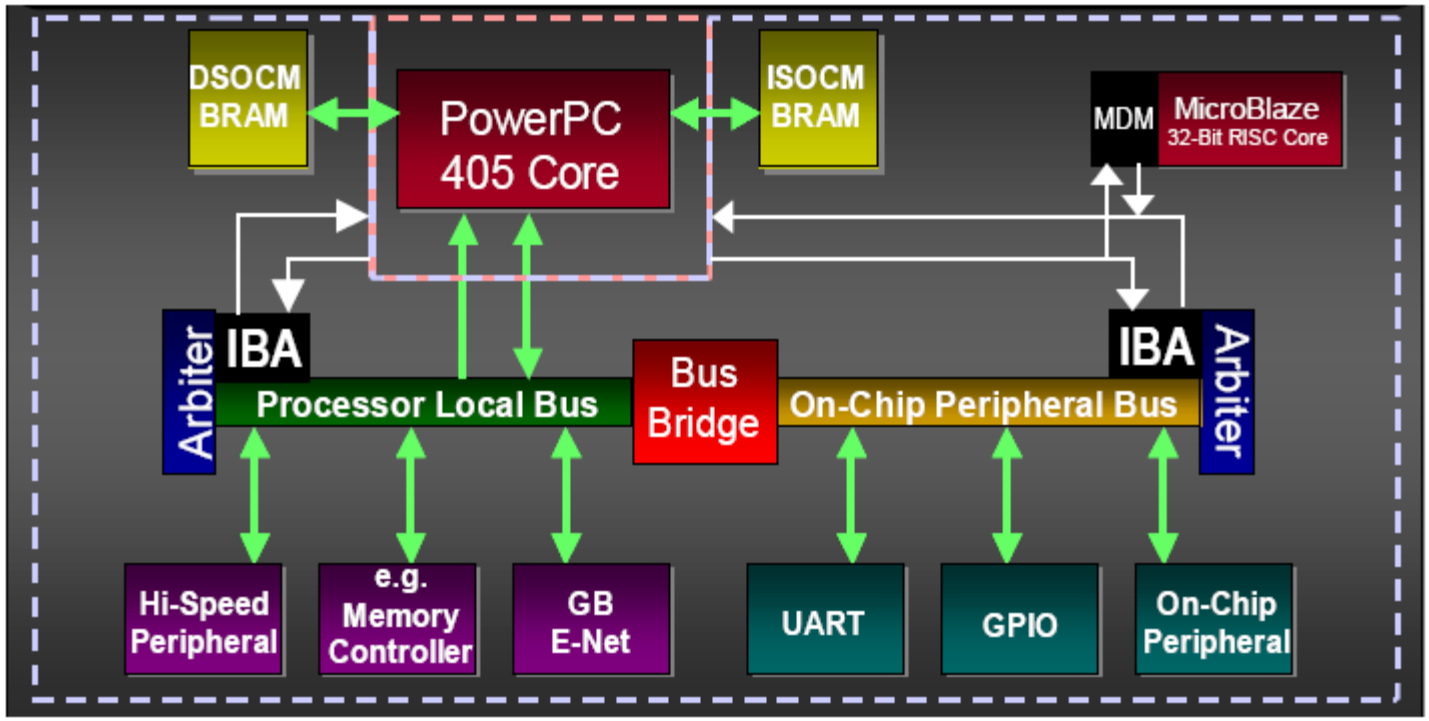


Logic analyzer architecture



Bus analyzer

- Monitors bus transaction for a specific bus (AHB/APB or PLB/OPB)

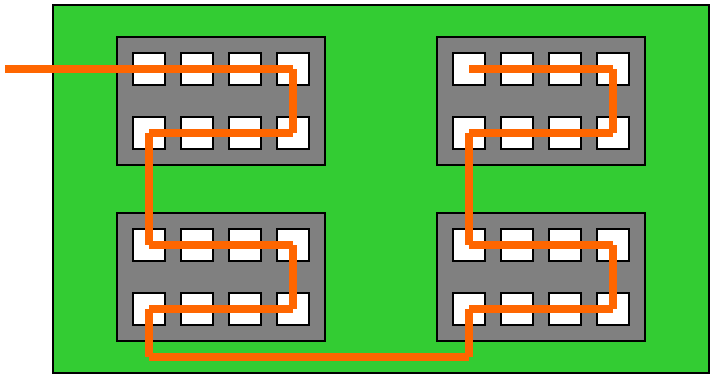


State and timing modes

- Timing mode: several samples per period
  - For glitch oriented debugging
  - more memory
- State mode: one sample per period
  - For sequential oriented problem

Boundary scan

- Simplifies testing of multiple chips on a board.
  - Registers on pins can be configured as a scan chain.
  - Used for debuggers, in-circuit emulators.
- JTAG



How to exercise code

- Run on host system.
- Run on target system.
- Run in instruction-level simulator.
- Run on cycle-accurate simulator.
- Run in hardware/software co-simulation environment.

Debugging real-time code

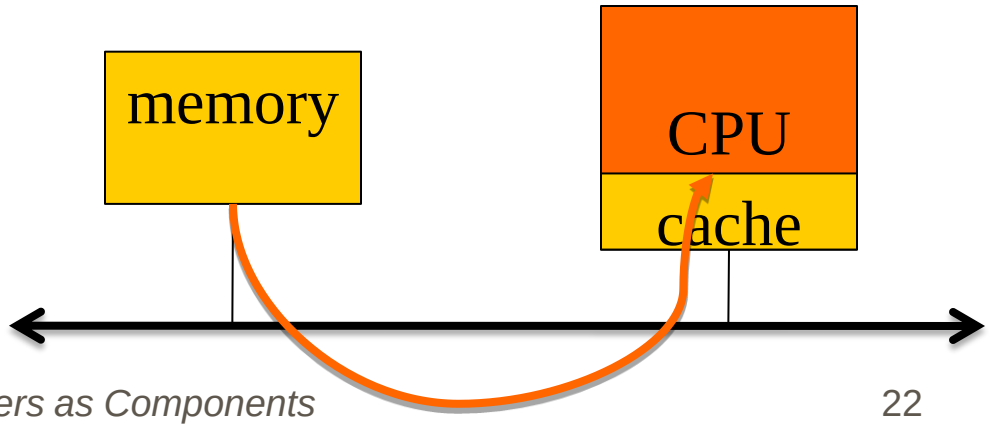
- Harder to diagnose
  - Bugs in drivers can cause non-deterministic behavior in the foreground problem.
  - Bugs may be timing-dependent.

4.7 System-level performance analysis

- Performance depends on all the elements of the system:
  - CPU.
  - Cache.
  - Bus.
  - Main memory.
  - I/O device.

Instruction fetch

- Move data from memory to CPU to process it
  - Read from memory.
  - Transfer over the bus to the cache
  - Transfer from the cache to CPU



Bandwidth as performance

- Bandwidth applies to several components:
  - Memory
  - Bus
  - CPU
- Different parts of the system run at different clock rates.
- Different components may have different widths (bus, memory).

Bandwidth and data transfers

- Per video frame: 320 x 240 x 3 = 230,400 bytes.
  - Transfer in 1/30 sec.
- Transfer 1 byte/μsec, 0.23 sec per frame.
  - Too slow.
- Increase bandwidth:
  - Increase bus width.
  - Increase bus clock rate.

H.264/AVC 720p

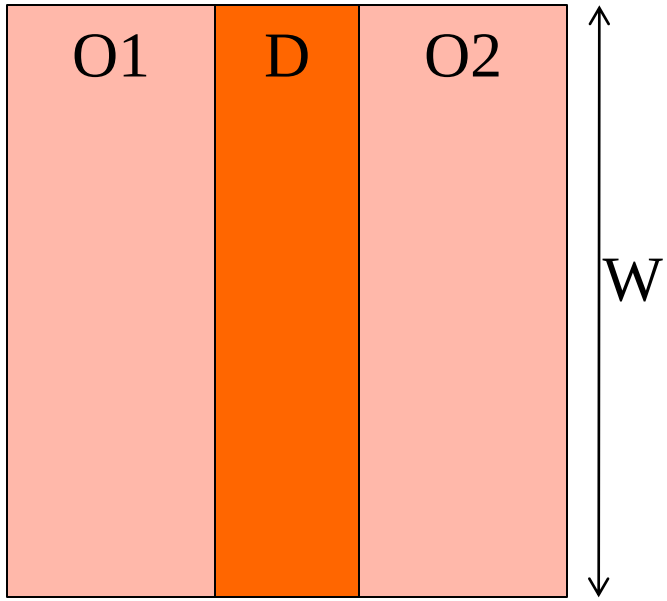
- One movie video without compression
  - 720 x 480 pixels per frame
  - 30 frames per second
  - Total 90 minutes
  - Full color

The total quantity of data = 167.96 G Bytes !!

- Video compression technique is important

Bus bandwidth

- T: # bus cycles.
- P: time/bus cycle.
- Total time for transfer:
  - t = TP.
- D: data payload length (cycles)
- O1 + O2 = overhead O (cycles)
  - Address, handshaking
- N bytes to be transferred
- Bus width: W bytes

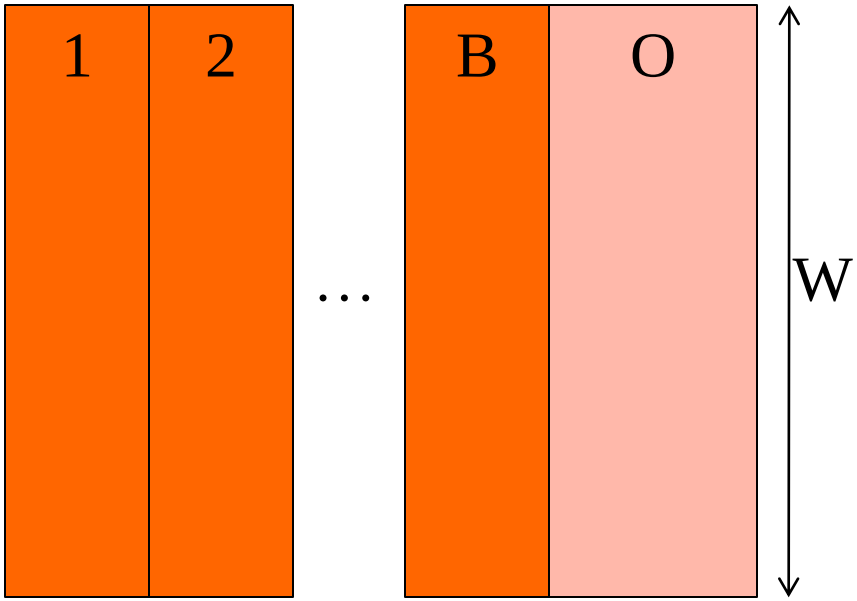


$$T_{basic}(N) = (D+O) \times N/W$$

Burst 횟수

Bus burst transfer bandwidth

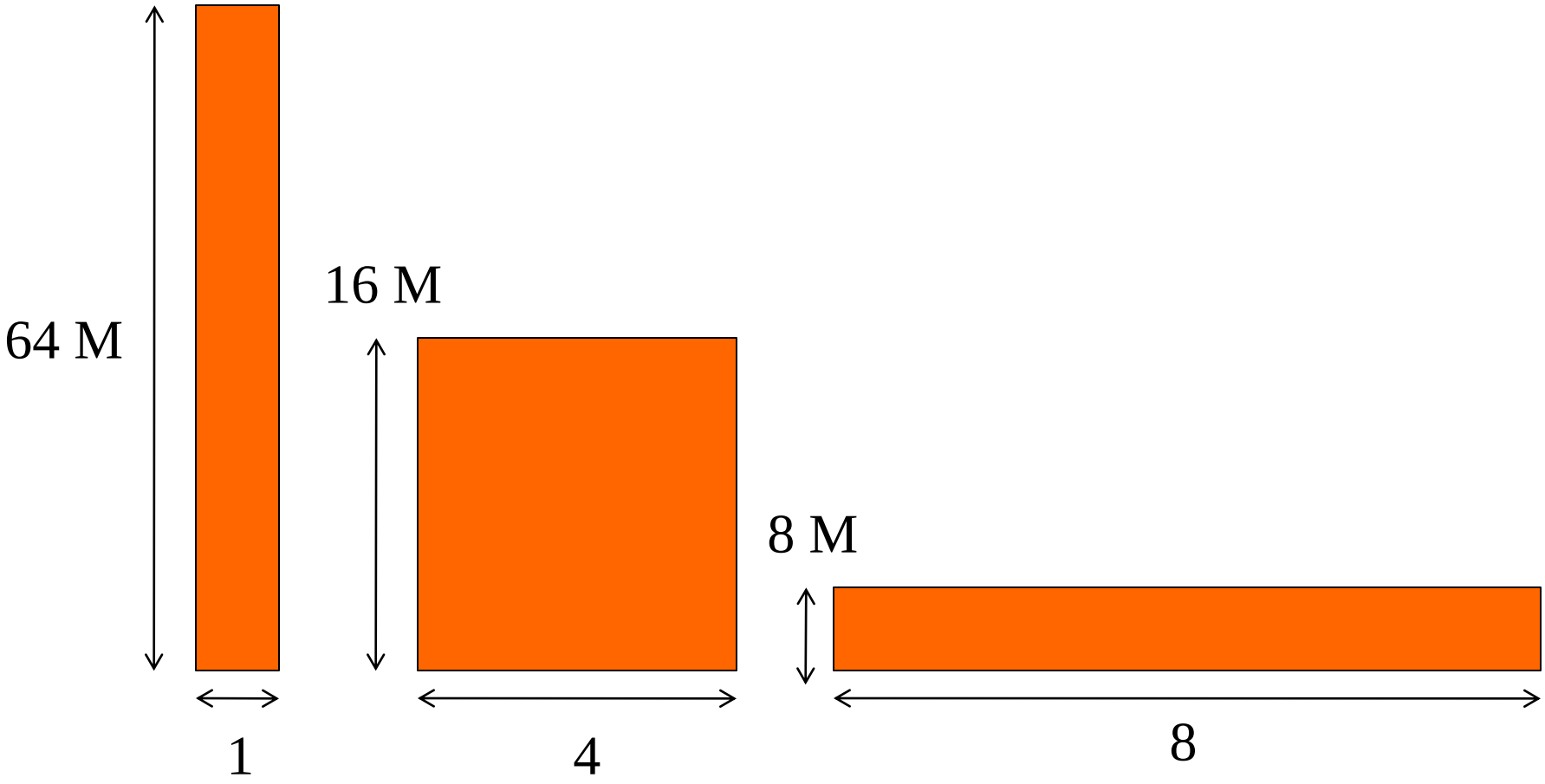
- T: # bus cycles.
- P: time/bus cycle.
- Total time for transfer:
  - t = TP.
- D: data payload length.
- O1 + O2 = overhead O.
- N bytes to be transferred



$$T_{burst}(N) = (BD+O) \times N/(BW)$$

Burst 횟수

Memory aspect ratios

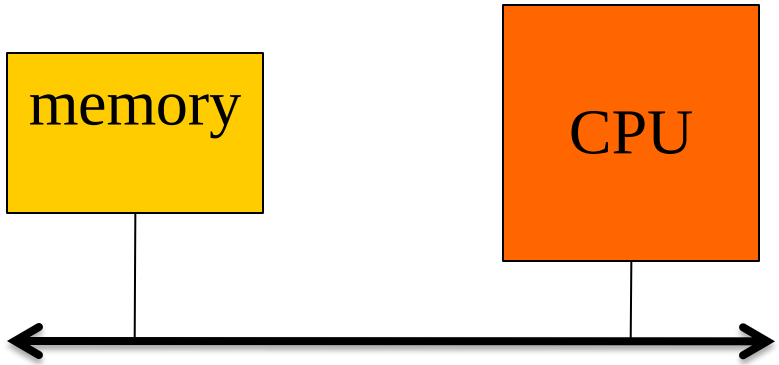


Memory access times

- Memory component access times comes from chip data sheet.
  - Page modes allow faster access for successive transfers on same page.
- What if data doesn't fit naturally into physical words:
- A pixel: RGB 24-bit
  - an access for 24-bit-wide memory
  - 3 accesses for 8-bit wide memory
  - how about 32-bit wide memory
    - waste one byte for each access
    - packing

Bus performance bottlenecks

- Transfer 320 x 240 video frame @ 30 frames/sec = 612,000 bytes/sec.
- Is performance bottleneck bus or memory?



Bus performance bottlenecks, cont'd.

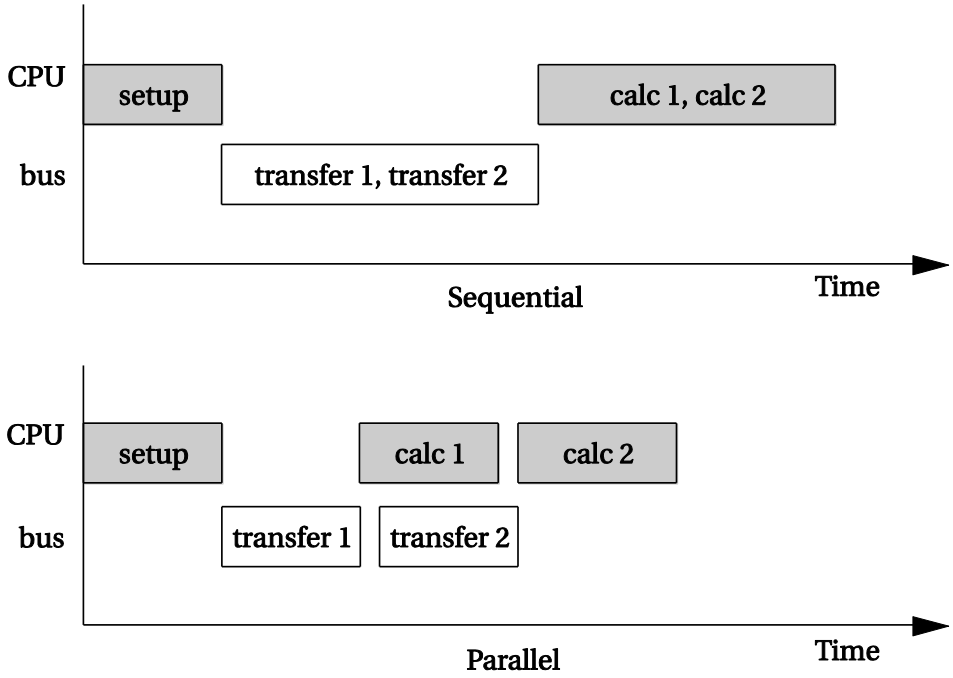
- Bus: assume 1 MHz bus, D=1, O=3:
  - T_{basic} = (1+3)612,000/2 = 1,224,000 cycles = 1.224 sec.
- Memory: try burst mode B=4, width w=0.5. (assume 10MHz)
  - T_{mem} = (4*1+4)612,000/(4*0.5) = 2,448,000 cycles = 0.2448 sec.

Performance spreadsheet

bus					memory				
clock period		1.00E-06			clock period		1.00E-08		
W		2			W		0.5		
D		1			D		1		
O		3			O		4		
					B		4		
N		612000			N		612000		
T_basic		1224000			T_mem		2448000		
t		1.22E+00			t		2.45E-02		

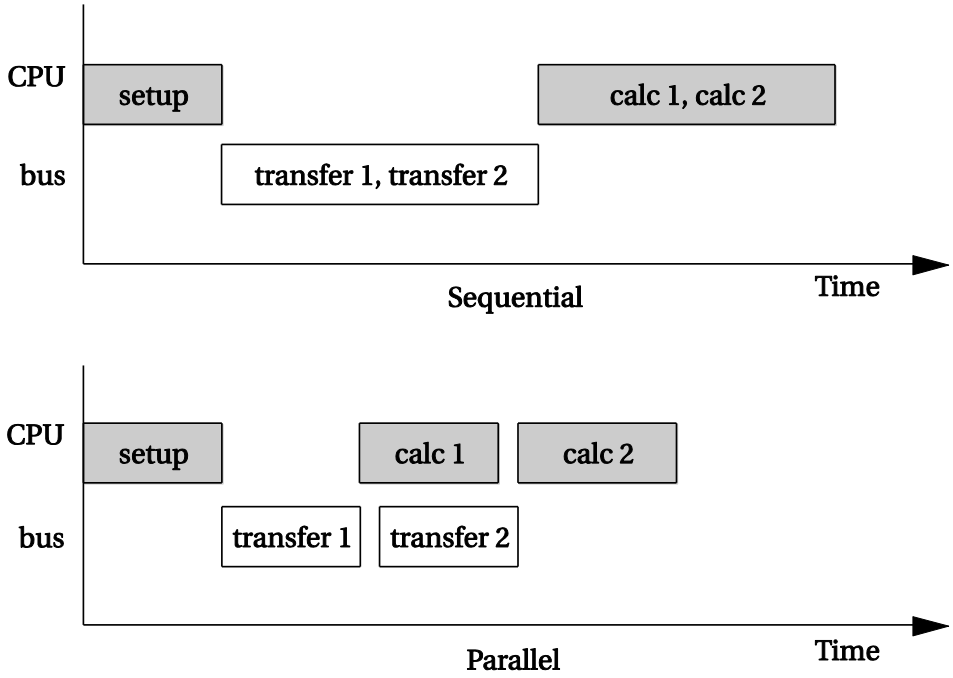
4.7.2 Parallelism

- Speed things up by running several units at once.
- DMA provides parallelism if CPU doesn't need the bus:
  - DMA + bus.
  - CPU.



4.7.2 Parallelism

- Speed things up by running several units at once.
- DMA provides parallelism if CPU doesn't need the bus:
  - DMA + bus.
  - CPU.



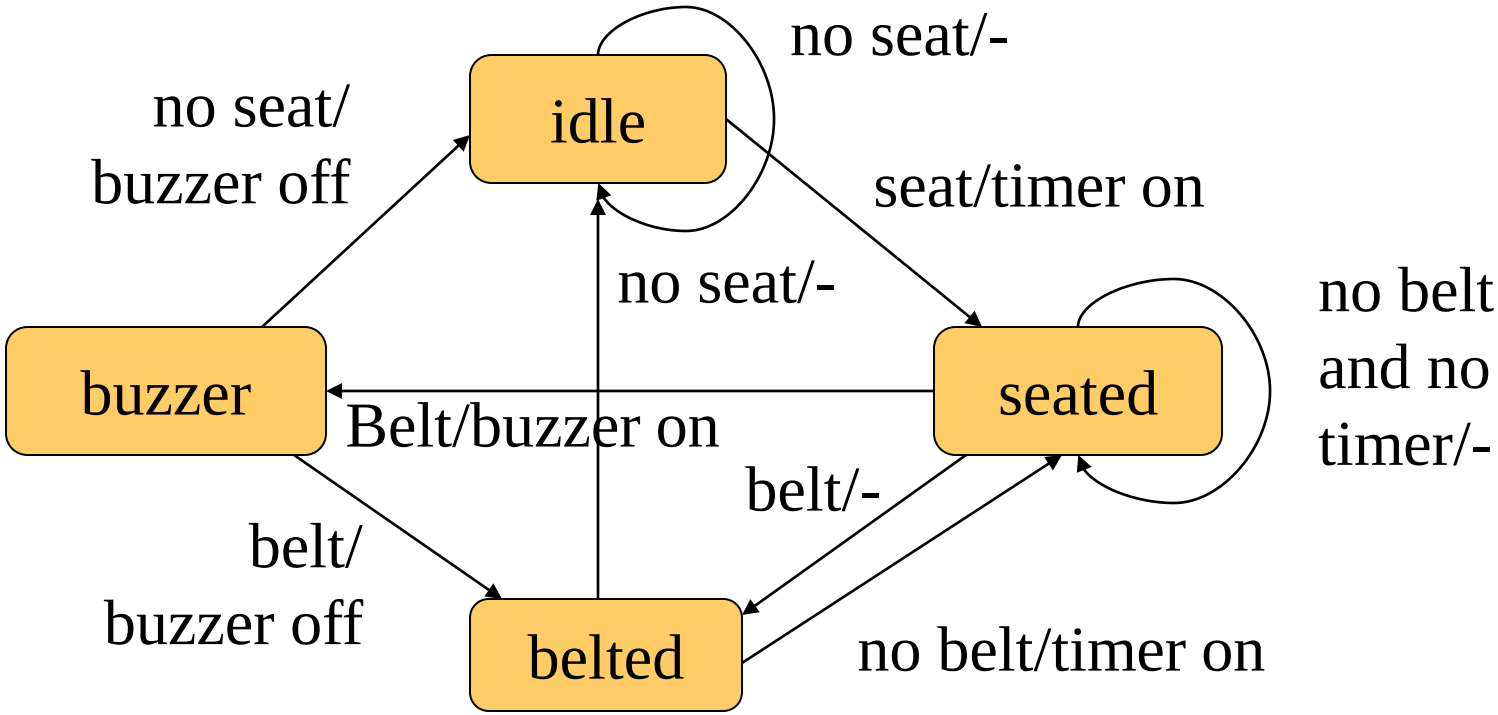
5. Program design and analysis

- Software components.
- Representations of programs
  - Data flow
  - Control flow
- Compilation
- Assembly and linking.

State machine

- Suitable to reactive systems
- Interactive
  - At their own speed
  - Making it wait
- Reactive
  - Intended to be deterministic
  - Cannot wait
- Real-time
  - value: logical correctness
  - when: timing constraints

State machine example

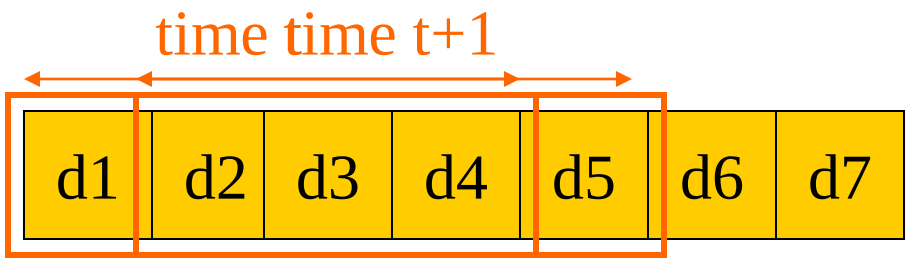


C implementation

```
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3
switch (state) {
    case IDLE: if (seat) { state = SEATED; timer_on = TRUE; }
            break;
    case SEATED: if (belt) state = BELTED;
                    else if (timer) state = BUZZER;
            break;
    …
}
```

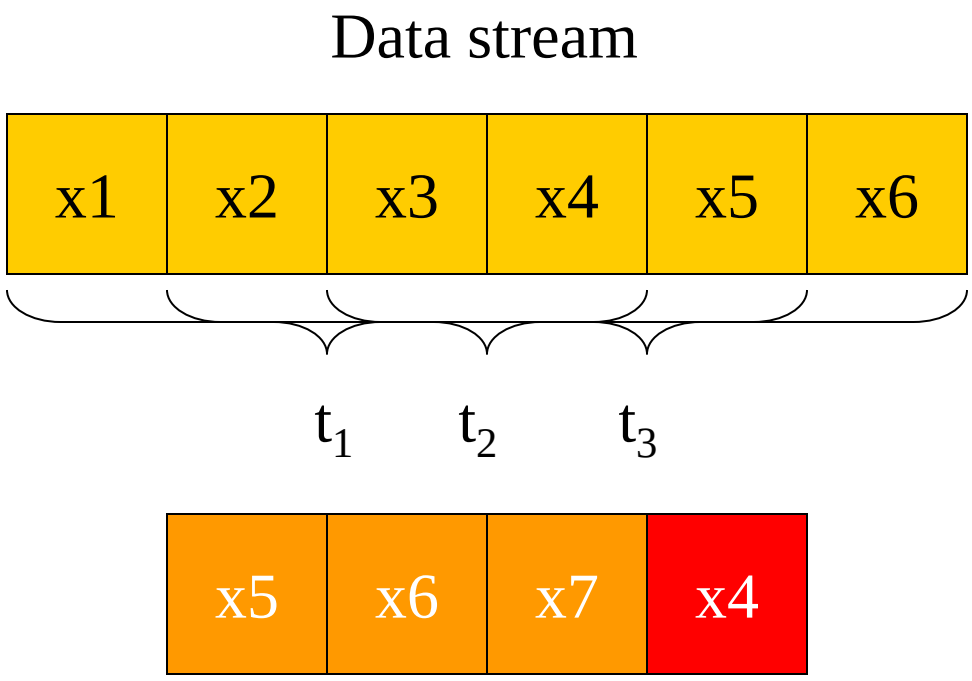
Signal processing and circular buffer

- Commonly used in signal processing:
  - new data constantly arrives;
  - each datum has a limited lifetime.



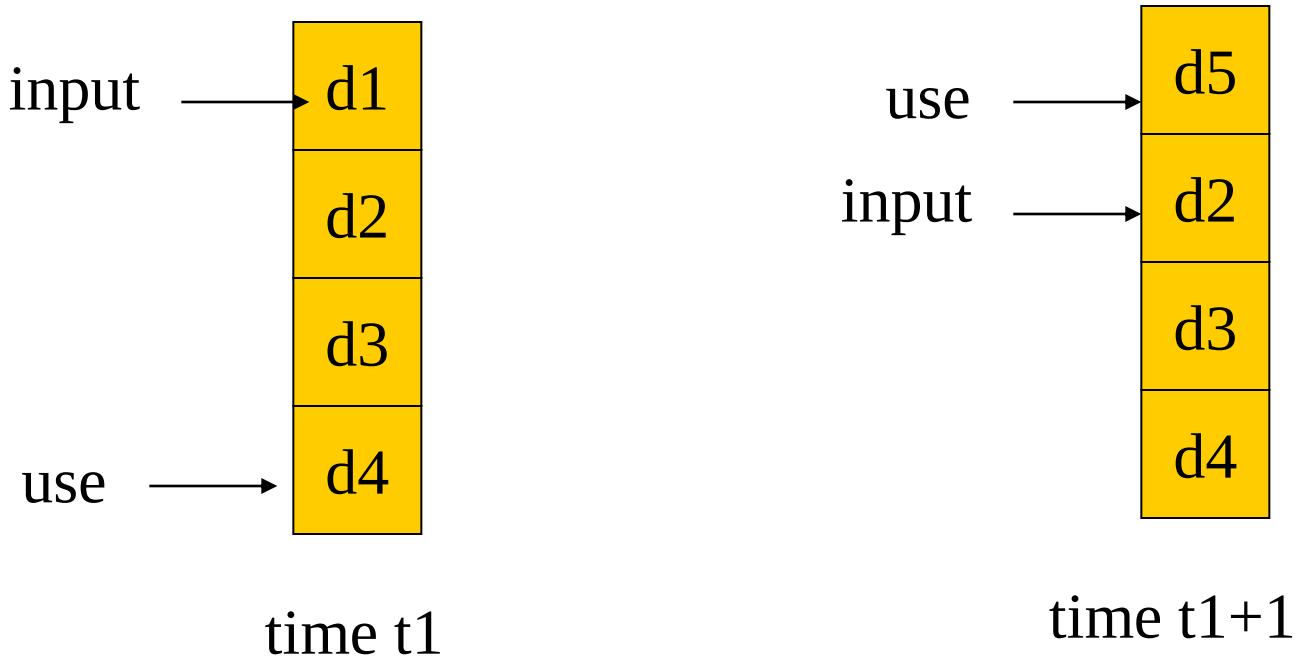
- Use a circular buffer to hold the data stream.

Circular buffer



Circular buffers

- Indexes locate currently used data, current input data:



Circular buffer implementation: FIR filter

```
int circ_buffer[N], circ_buffer_head = 0;
int c[N]; /* coefficients */
…
int ibuf, ic;
for (f=0, ibuff=circ_buff_head, ic=0;
    ic<N; ibuff=(ibuff==N-1?0:ibuff++), ic++)
f = f + c[ic]*circ_buffer[ibuf];
```

Queues

- Elastic buffer: holds data that arrives irregularly.

Buffer-based queues

```
#define Q_SIZE 32
#define Q_MAX (Q_SIZE-1)
int q[Q_MAX], head, tail;
void initialize_queue() { head =
    tail = 0; }
void enqueue(int val) {
    if (((tail+1)%Q_SIZE) ==
    head) error();
    q[tail]=val;
    if (tail == Q_MAX) tail = 0;
    else tail++;
}
```

```
int dequeue() {
    int returnval;
    if (head == tail) error();
    returnval = q[head];
    if (head == Q_MAX) head =
    0;
        else head++;
    return returnval;
}
```

5.2 Models of programs

- Source code is not a good representation for programs:
  - clumsy;
  - leaves much information implicit.
- Compilers derive intermediate representations to manipulate and optimize the program.

Data flow graph

- DFG: data flow graph.
- Does not represent control.
  - No conditional
- Models basic block:
  - code with an entry and an exit.
- Describes the minimal ordering requirements on operations.

Single assignment form

```
w = a + b;
x =  a - c;
y = x + d;
x = a + c;
z = y + e;
```

original basic block

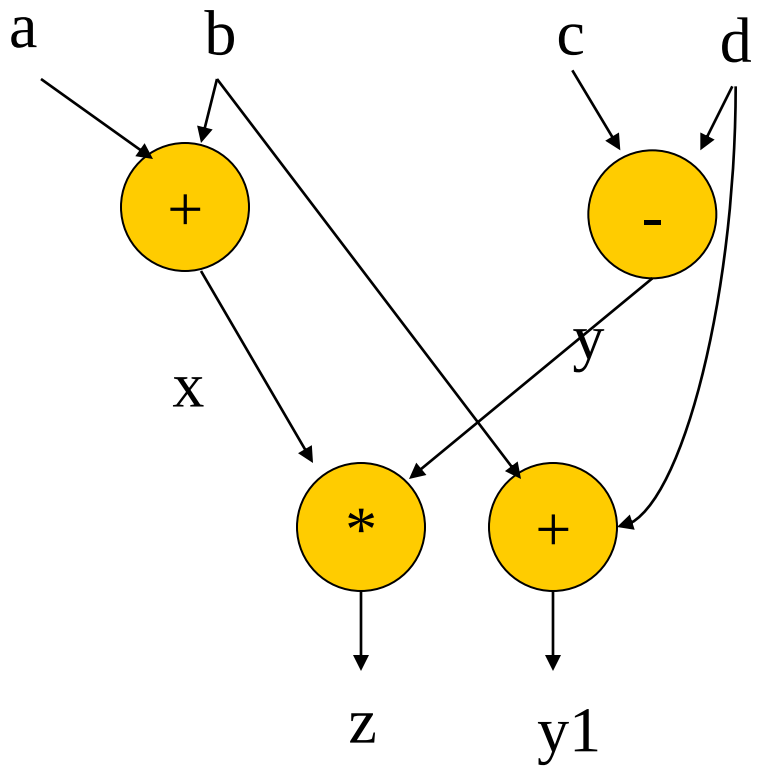
```
w = a + b;
x1 = a - c;
y = x1+ d;
x2 = a + c;
z = y + e;
```

single assignment form

Data flow graph

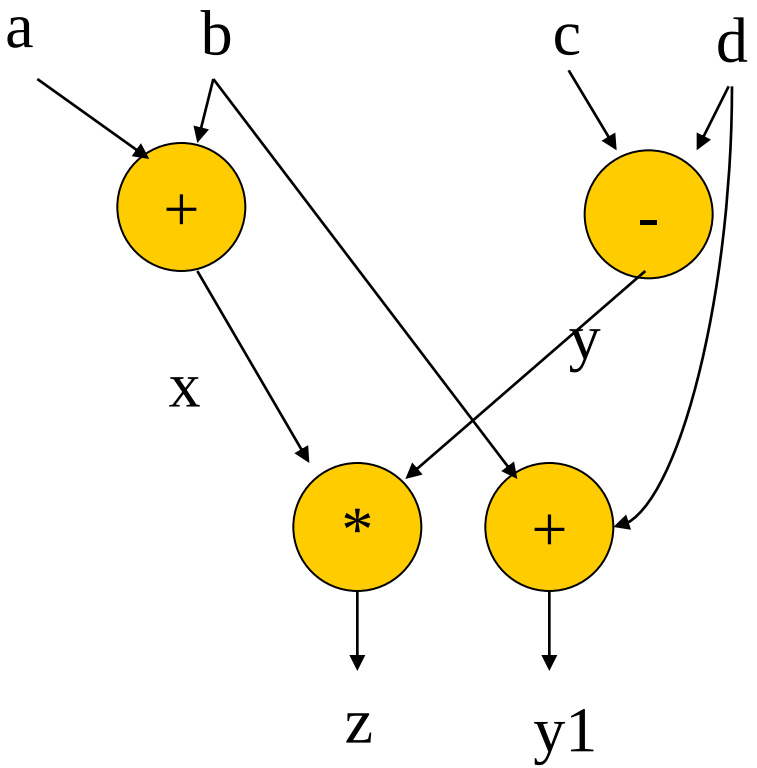
```
x = a + b;
y = c - d;
z = x * y;
y1 = b + d;
```

First, convert it to single assignment form



DFG

DFGs and partial orders



Partial order:

- a+b, c-d; b+d, x*y

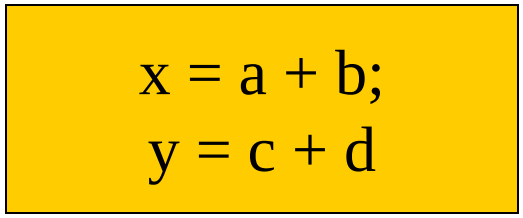
Can do pairs of operations in any order.

Control-data flow graph

- CDFG: represents control and data.
- Uses data flow graphs as components.
- Two types of nodes:
  - decision;
  - data flow.

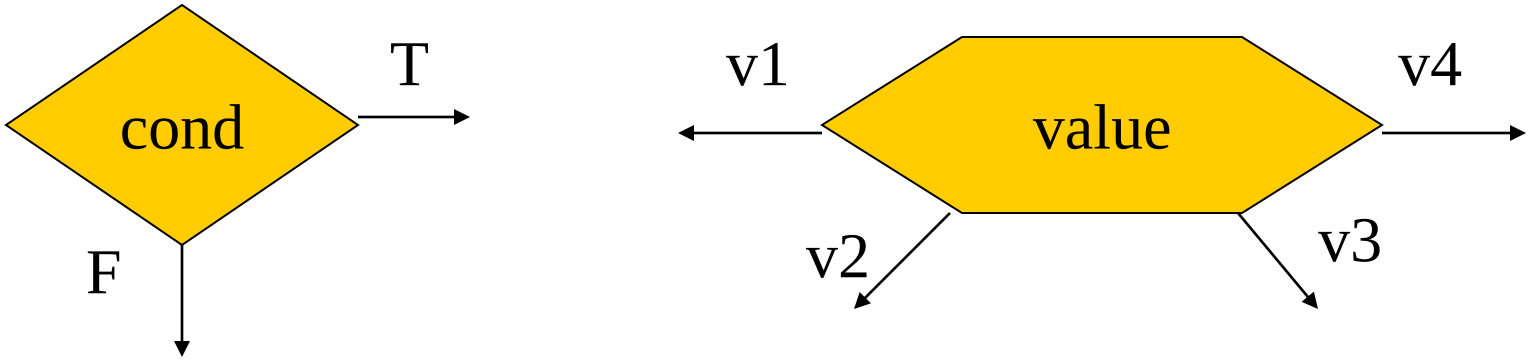
Data flow node

Encapsulates a data flow graph:



Write operations in basic block form for simplicity.

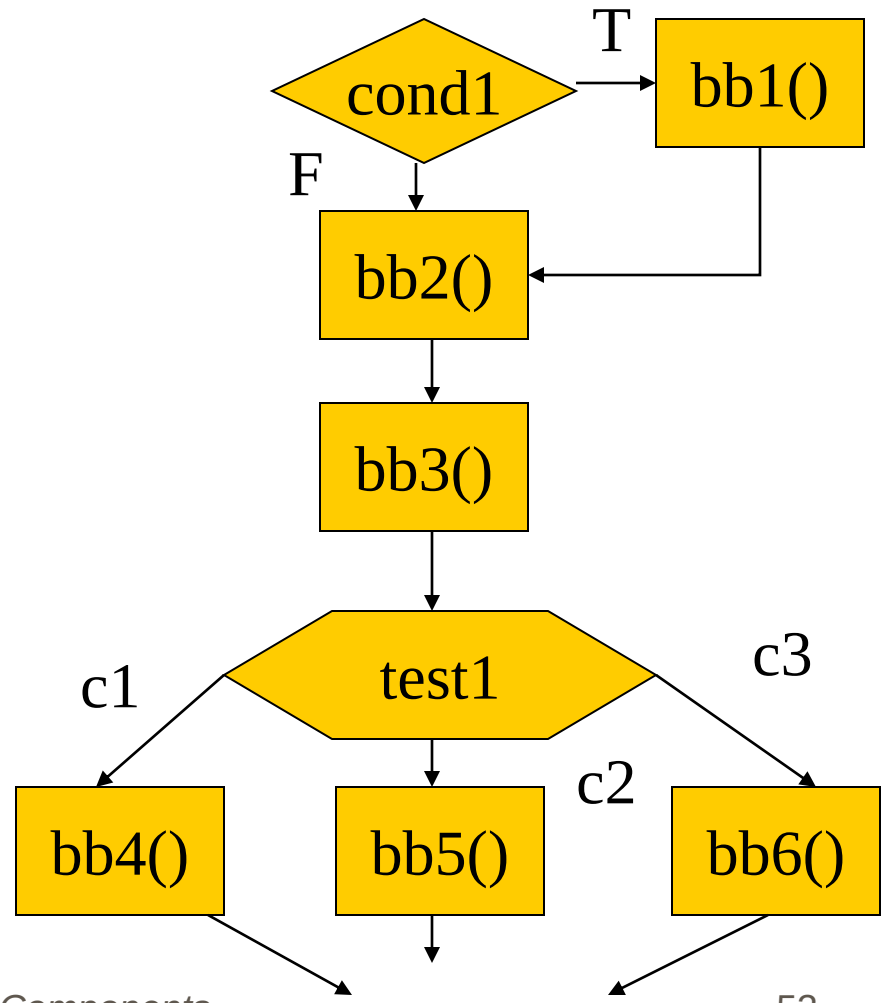
Control



Equivalent forms

CDFG example

```
if (cond1) bb1();
 else bb2();
bb3();
switch (test1) {
    case c1: bb4(); break;
    case c2: bb5(); break;
    case c3: bb6(); break;
}
```



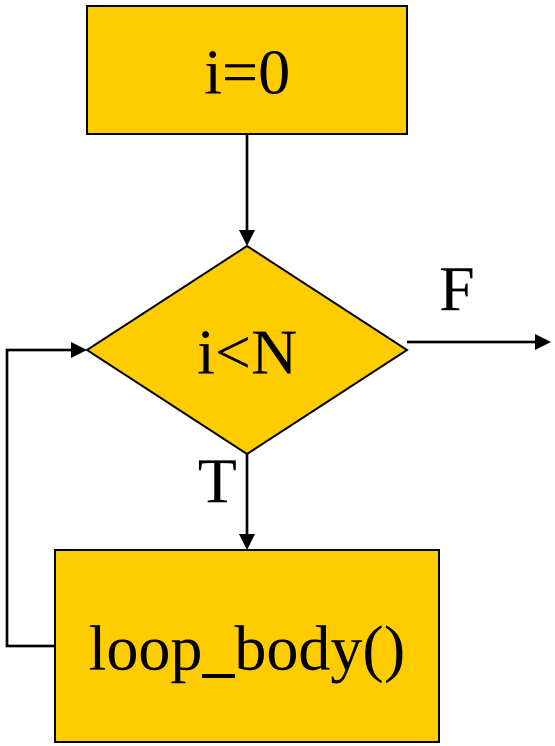
for loop

```
for (i=0; i<N; i++)
   loop_body();
```

for loop

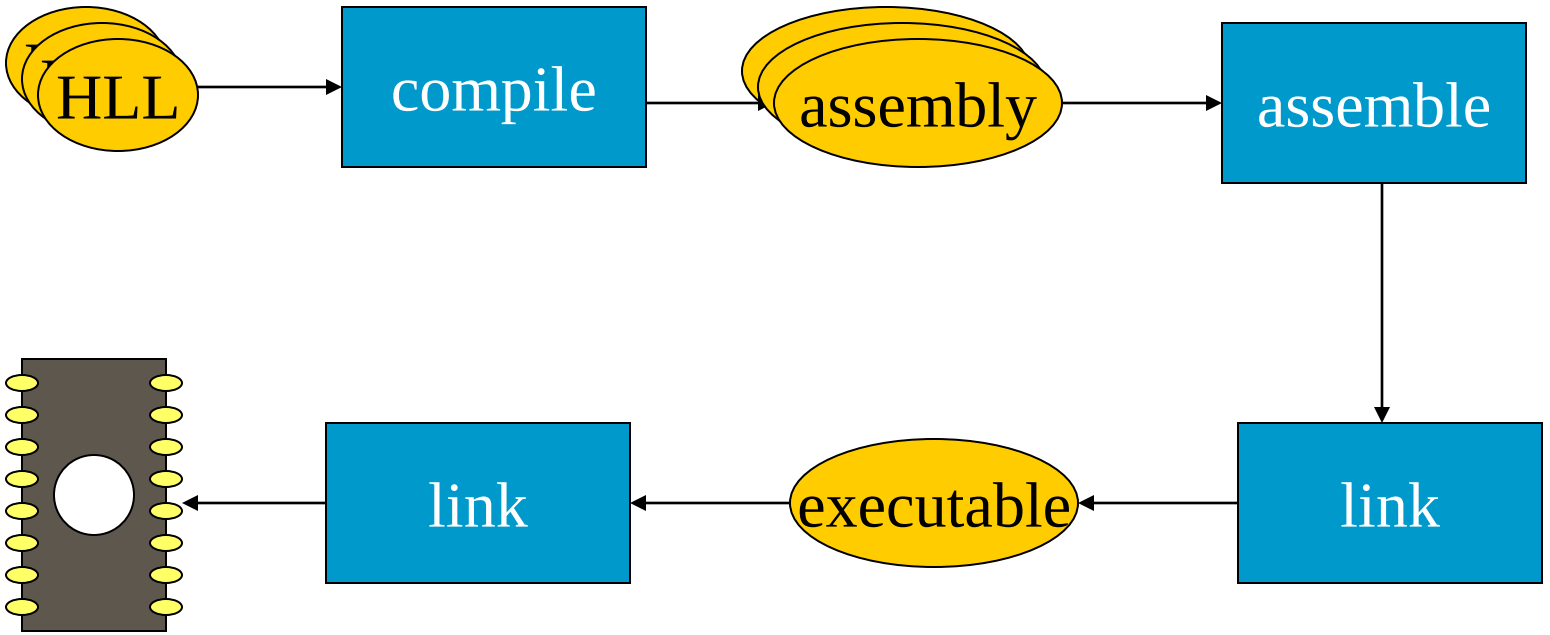
```
i=0;
while (i<N) {
   loop_body(); i++; }
```

equivalent

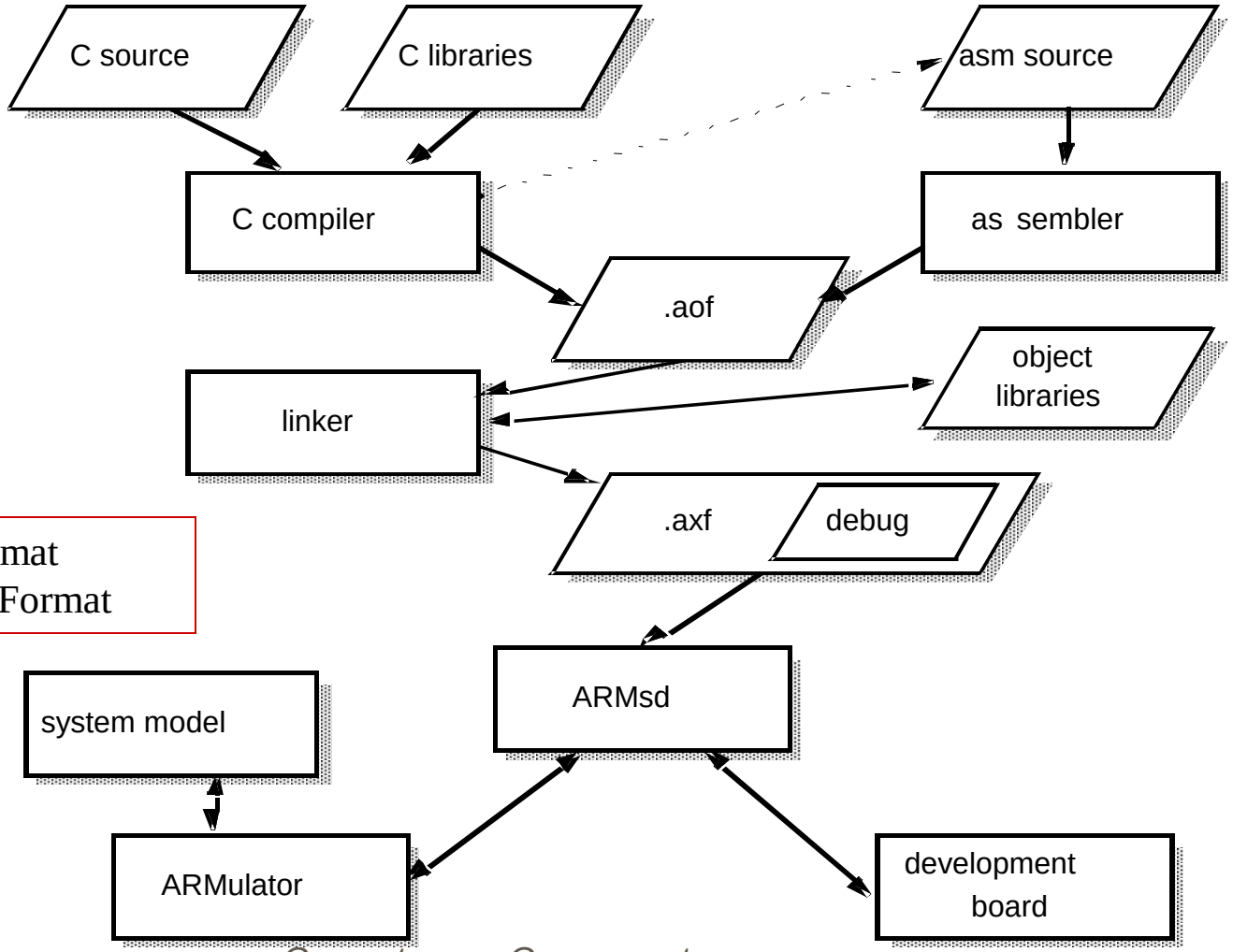


Assembly and linking

- Last steps in compilation:



ARM Development tools



aof - ARM Object Format
axf - ARM Executive Format

Multiple-module programs

- Programs may be composed from several files.
- Addresses become more specific during processing:
  - relative addresses are measured relative to the start of a module;
  - absolute addresses are measured relative to the start of the CPU address space.

Assemblers

- Major tasks:
  - generate binary for symbolic instructions;
  - translate labels into addresses;
  - handle pseudo-ops (data, etc.).
- Generally one-to-one translation.
- Assembly labels:

```
            ORG 100
label1   ADR r4,c
```

Symbol table

```
        ADD r0,r1,r2
xx      ADD r3,r4,r5
        CMP r0,r3
yy      SUB r5,r6,r7
```

assembly code

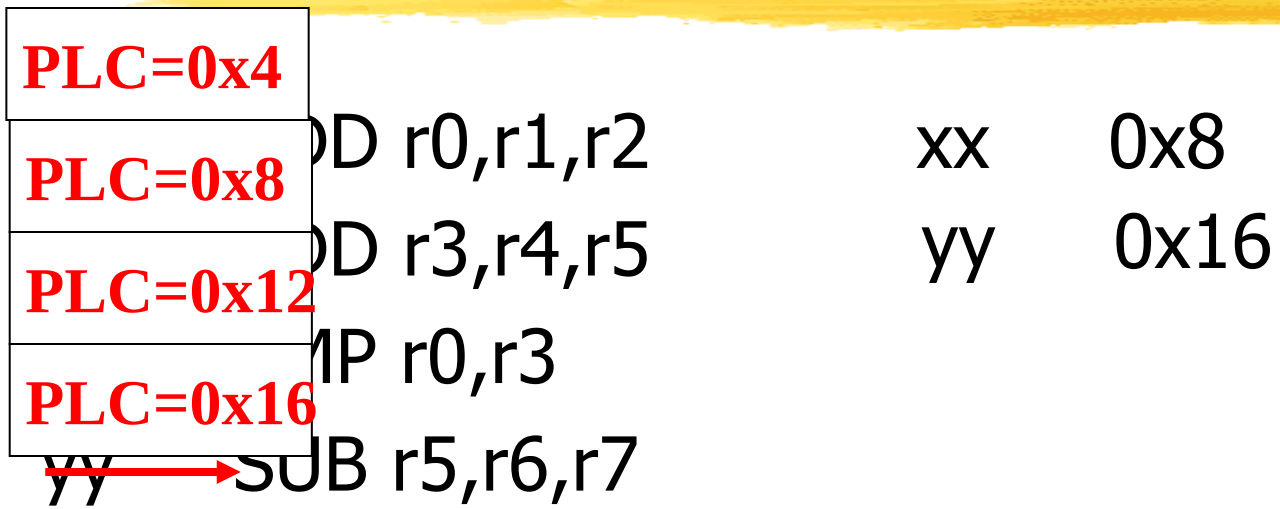
xx	0x8
yy	0x10

symbol table

Symbol table generation

- Use program location counter (PLC) to determine address of each location.
- Scan program, keeping count of PLC.
- Addresses are generated at assembly time, not execution time.

Symbol table example



Two-pass assembly

- Pass 1:
  - generate symbol table
- Pass 2:
  - generate binary instructions

Relative address generation

- Some label values may not be known at assembly time.
- Labels within the module may be kept in relative form.
- Must keep track of external labels---can't generate full binary for instructions that use external labels.

Pseudo-operations

- Pseudo-ops do not generate instructions:
  - ORG sets program location.
  - EQU generates symbol table entry without advancing PLC.
  - Data statements define data blocks.

Linking

- Combines several object modules into a single executable module.
- Jobs:
  - put modules in order;
  - resolve labels across modules.

Externals and entry points

- Externals:
- Entry points
- Combines several object modules into a single executable module.
- Jobs:
  - put modules in order;
  - resolve labels across modules.

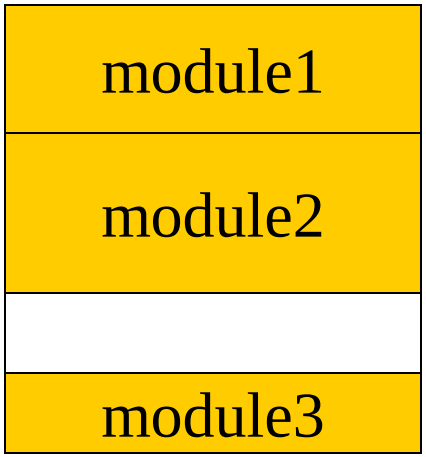
Externals and entry points

```
// file 1
label1   LDR r0,[r1]
         …
         ADR a
         …
         B label2
         …
var1      %1

----------------
External ref        entry points
a                   label1
label2              var1
```

```
//file2
label2   ADR var1
         …
         B label3
         …
x         %1
y         %1
a         %10

----------------
External ref        entry points
var1                label2
label3              x
                    y
                    a
```

Module ordering

- Code modules must be placed in absolute positions in the memory space.
- Load map or linker flags control the order of modules.



Dynamic linking

- Some operating systems link modules dynamically at run time:
  - shares one copy of library among all executing programs;
  - allows programs to be updated with new versions of libraries.