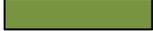
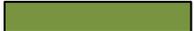


Memory Systems

Memory Needs

Function	Description	Volatility	Size	Speed	Access	Portable	Technology
Data	Temp data/operand store for programs	V			Direct	N	DDR/SRAM
Instructions	Opcode storage for executing programs	V/NV			Direct	Y/N	DDR/SRAM/Flash
Parameters	BIOS settings, configurations	NV			Direct	N	EEPROM/Flash
OS/System	System and user programs	NV			Indirect	N	HD/SSD/SD/Flash
Data file store	Audio, video, other mass-store data	NV			Indirect	Y/N	HD/SSD/SD/Flash

Fast memory busses (like DDR) use different physical pin drivers than regular busses

Large-array memory busses (like HD) use high-speed serial busses like SATA or PCIe

Dynamic memory (like DDR) has much smaller cell size and much higher density, but must be refreshed

Main types of memory

Volatile (RAM): DDR, SRAM, PSRAM, Cellular RAM

Dynamic vs. Static

Dynamic: DDR, PSRAM, Cellular Static: SRAM

High-speed vs. regular

High speed: > 200MHz

Serial vs. parallel

Multiplexed

Multiplexed port vs. Multiplexed part

Synchronous/piped/burst

High throughput – device changes its own address

Non-volatile (ROM): ROM, EEPROM, Flash, HD (magnetic)

Serial vs. parallel

In-system vs. Removable

Flash vs. EEPROM

Mass-store (magnetic and solid-state)

Micron: Leading supplier of RAM and ROM

Where's the SRAM?

Memory

DRAM

DDR4 SDRAM

DDR3 SDRAM

DDR2 SDRAM

DDR SDRAM

SDRAM

GDDR

RLDRAM Memory

LPDRAM

DRAM Modules

RDIMM

VLP RDIMM

VLP UDIMM

UDIMM

SODIMM

SORDIMM

VLP Mini-DIMM

LRDIMM

Mini-DIMM

NVDIMM

NAND Flash

3D NAND

TLC NAND

MLC NAND

SLC NAND

Managed NAND

e-MMC

Embedded USB

Universal Flash Storage

NOR Flash

Parallel NOR Flash

Serial NOR Flash

Xccela™ Flash

Hybrid Memory Cube

Short-Reach HMC

Multichip Packages

UFS-Based MCP

e.MMC-Based MCP

NAND-Based MCP

NOR-Based MCP

Micron: Leading supplier of RAM and ROM

Where's the SRAM?



128Mb: x8/x16, 3V, MT28EW Embedded Parallel NOR
Read AC Characteristics

Read AC Characteristics

Table 30: Read AC Characteristics – $V_{CC} = V_{CCQ} = 2.7-3.6V$

Parameter	Symbol		Condition	Min	Max	Unit	Notes
	Legacy	JEDEC					
Address valid to next address valid	t^1_{RC}	t^1_{AVAV}	$CE\# = V_{IL}$, $OE\# = V_{IL}$	70	–	ns	
Address valid to output valid	t^1_{ACC}	t^1_{AVQV}	$CE\# = V_{IL}$, $OE\# = V_{IL}$	–	70	ns	
Address valid to output valid (page)	t^1_{PAGE}	t^1_{AVQV1}	$CE\# = V_{IL}$, $OE\# = V_{IL}$	–	20	ns	
CE# LOW to output valid	t^1_{CE}	t^1_{ELQV}	$OE\# = V_{IL}$	–	70	ns	
OE# LOW to output valid	t^1_{OE}	t^1_{GLQV}	$CE\# = V_{IL}$	–	25	ns	
CE# HIGH to output High-Z	t^1_{HZ}	t^1_{EHQZ}	$OE\# = V_{IL}$	–	20	ns	1
OE# HIGH to output High-Z	t^1_{DF}	t^1_{GHQZ}	$CE\# = V_{IL}$	–	15	ns	1
CE# HIGH, OE# HIGH, or address transition to output transition	t^1_{OH}	t^1_{EHQX} , t^1_{GHQX} , t^1_{AXQX}	–	0	–	ns	
CE# LOW to BYTE# LOW	t^1_{ELFL}	t^1_{ELBL}	–	–	10	ns	
CE# LOW to BYTE# HIGH	t^1_{ELFH}	t^1_{ELBH}	–	–	10	ns	
BYTE# LOW to output valid	t^1_{FLQV}	t^1_{BLQV}	–	–	1	μs	
BYTE# HIGH to output valid	t^1_{FHQV}	t^1_{BHQV}	–	–	1	μs	

Memory

DRAM

DDR4 SDRAM

DDR3 SDRAM

DDR2 SDRAM

DDR SDRAM

SDRAM

GDDR

RLDRAM Memory

LPDRAM

DRAM Modules

RDIMM

VLP RDIMM

VLP UDIMM

UDIMM

SODIMM

SORDIMM

VLP Mini-DIMM

LRDIMM

Mini-DIMM

NVDIMM

NAND Flash

3D NAND

TLC NAND

MLC NAND

SLC NAND

Managed NAND

e-MMC

Embedded USB

Universal Flash Storage

NOR Flash

Parallel NOR Flash

Serial NOR Flash

Xccela™ Flash

Hybrid Memory Cube

Short-Reach HMC

Multichip Packages

UFS-Based MCP

e.MMC-Based MCP

NAND-Based MCP

NOR-Based MCP

ISSI: Leading supplier of non-leading-edge RAM and ROM

DRAM

DDR4 SDRAM
DDR3 SDRAM
DDR3 SDRAM w/ ECC
DDR2 SDRAM
DDR SDRAM
SDR SDRAM
EDO & Fast Page Mode DRAM
RLDRAM® 2/3

Mobile DRAM

LPDDR2 SDRAM
Mobile DDR SDRAM
Mobile/Low Voltage SDR SDRAM

SRAM

Asynchronous SRAM
Serial SRAM & Low Pin Count SRAM
Synchronous SRAM
QUAD/QUADP & DDR-II/DDR-IIP
CellularRAM/Pseudo SRAM
HyperRAM™

MCP (Multi-Chip Package)

LPDDR2 DRAM + Serial NOR Flash

Wafer Level Memory Solutions

Wafer Level Memory Solutions

Analog

Audio Amplifiers
FxLED Driver
Backlight Driver
HBLED Driver
I/O Expanders
Sensor & Power Management

Automotive Analog

Flash

Serial NOR Flash
Serial NOR Flash w/ECC
Twin Serial NOR Flash
HyperFlash™
Parallel NOR Flash
SPI NAND Flash
NAND Flash
eMMC
Programmer Support
Flash Application Notes

ISSI: Leading supplier of non-leading-edge RAM and ROM

IS61WV102416FALL
IS61/64WV102416FBLL



AC CHARACTERISTICS (OVER OPERATING RANGE)

READ CYCLE AC CHARACTERISTICS

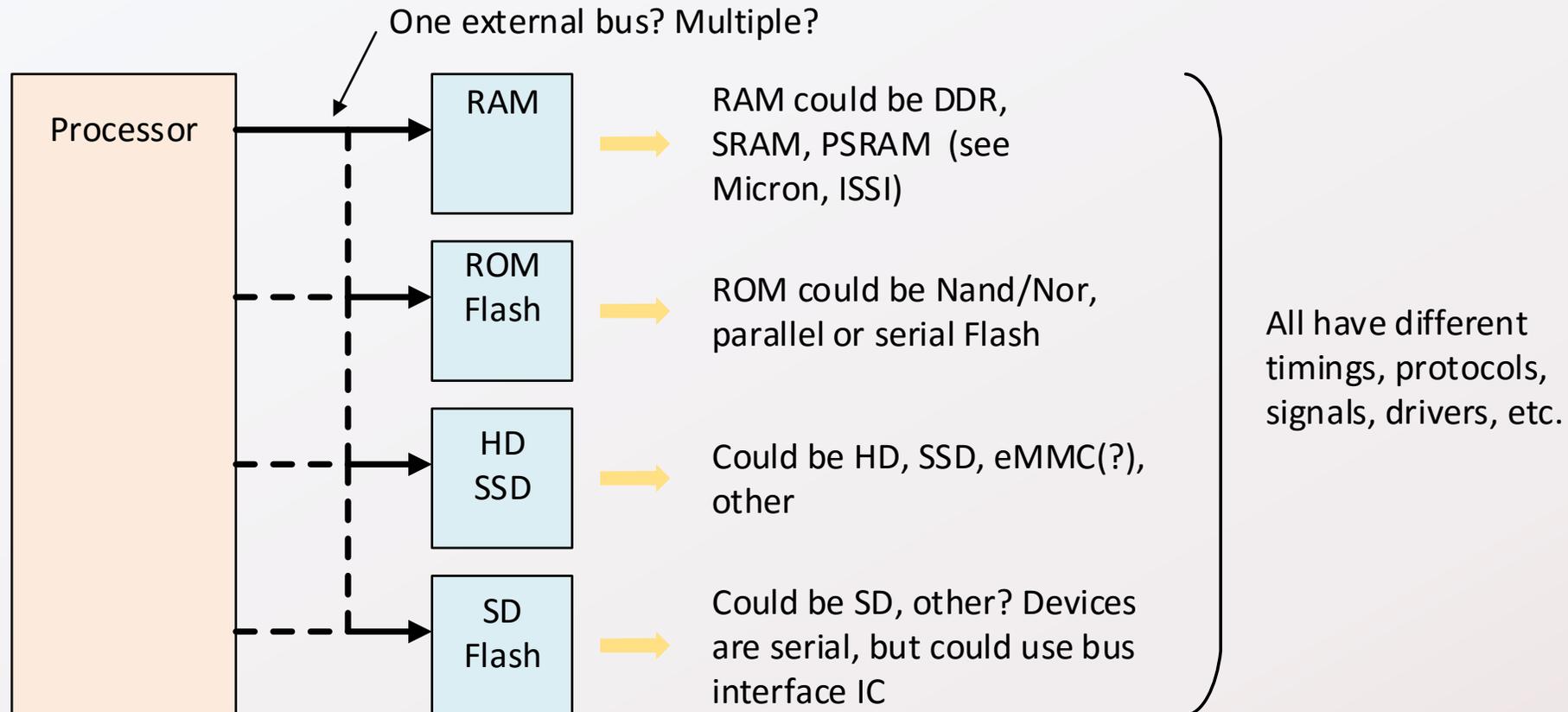
Parameter	Symbol	-8 ⁽¹⁾		-10 ⁽¹⁾		-20 ⁽¹⁾		unit	notes
		Min	Min	Min	Max	Min	Max		
Read Cycle Time	tRC	8	-	10	-	20	-	ns	
Address Access Time	tAA	-	8	-	10	-	20	ns	
Output Hold Time	tOHA	2.5	-	2.5	-	2.5	-	ns	
CS# Access Time	tACE	-	8	-	10	-	20	ns	
OE# Access Time	tDOE	-	5.5	-	6	-	8	ns	
OE# to High-Z Output	tHZOE	0	4	0	5	0	8	ns	2
OE# to Low-Z Output	tLZOE	0	-	0	-	0	-	ns	2
CS# to High-Z Output	tHZCE	0	4	0	5	0	8	ns	2
CS# to Low-Z Output	tLZCE	3	-	3	-	3	-	ns	2
UB#, LB# Access Time	tBA	-	5.5	-	6	-	8	ns	
UB#, LB# to High-Z Output	tHZB	0	4	0	5	0	8	ns	2
UB#, LB# to Low-Z Output	tLZB	0	-	0	-	0	-	ns	2

Bus partitions

Typically one physical bus, but many different potential devices/technologies.

Different devices have different timing, control, drive levels, etc

Different address ranges on same bus may need to behave differently



External Memory Interface (Busses)

Should processor bus be designed for:

DDR? SRAM? What speed? What width?

Can bus accommodate different devices or technologies?

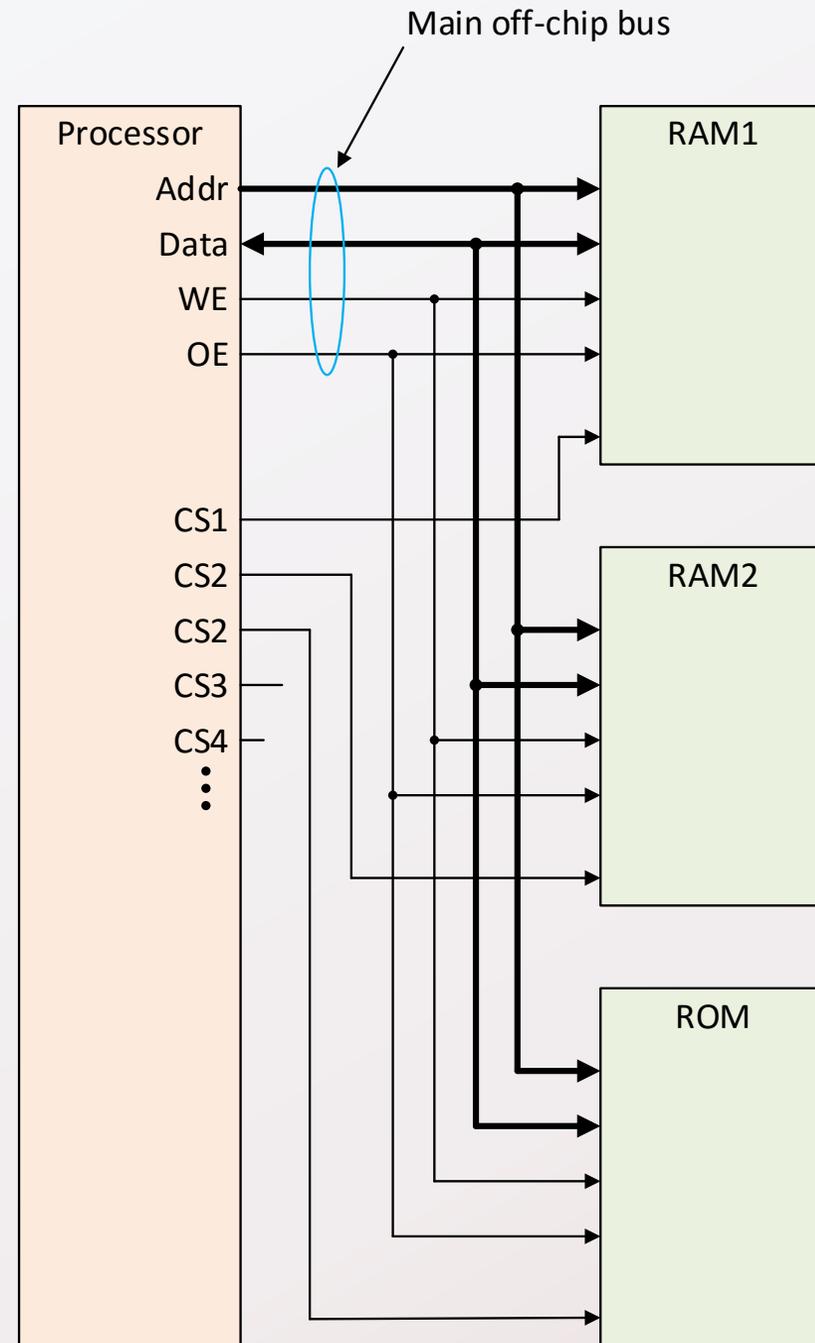
Different access speeds?

Bus Grant/Bus acknowledge

Bus hold-off (dynamic wait state)

Different control signals?

Different signal drive/protocols?



ZYNQ

External busses typically target one type of memory; if more flexibility needed, more busses used

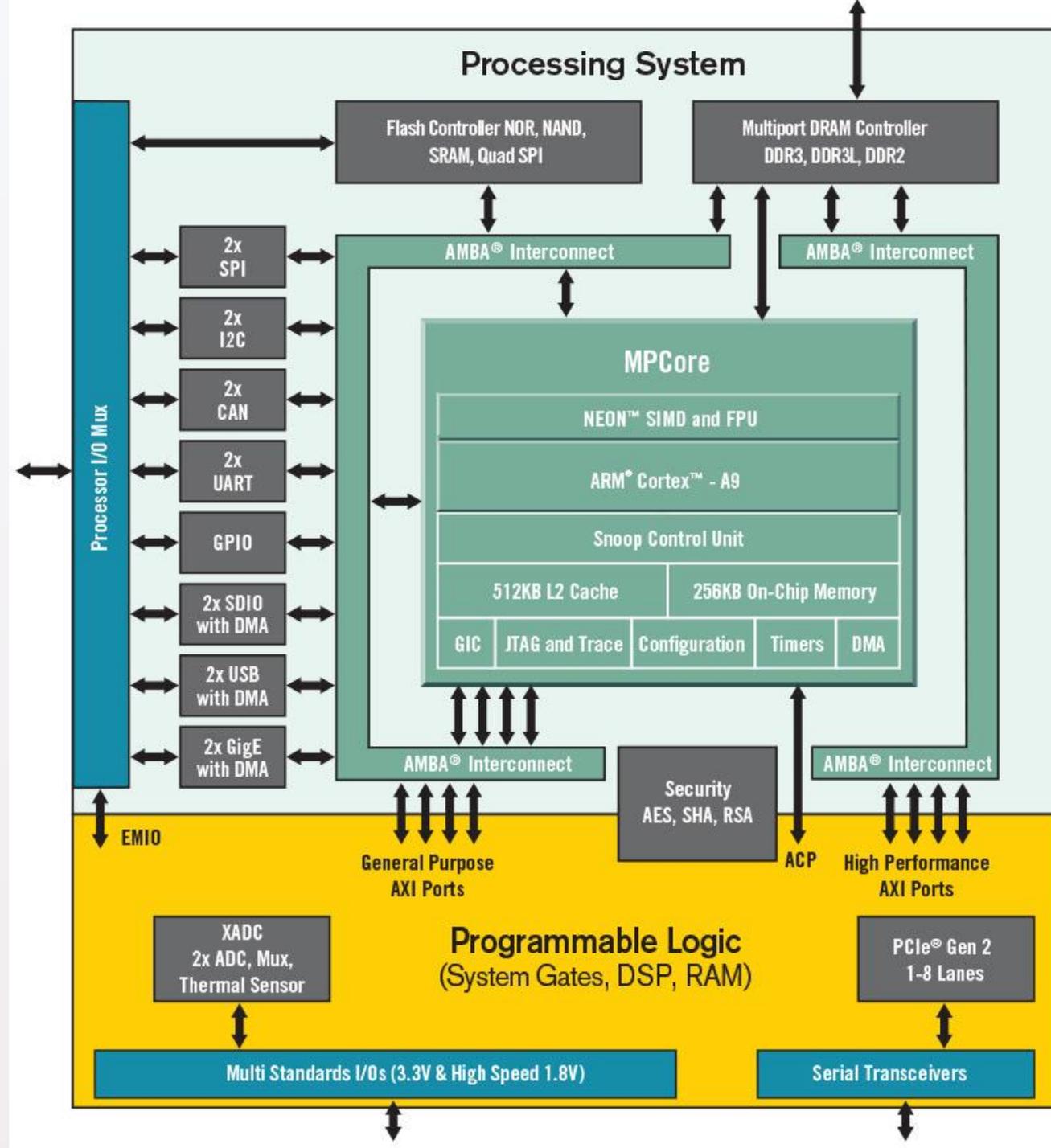


Figure 4. STM32F427xx and STM32F429xx block diagram

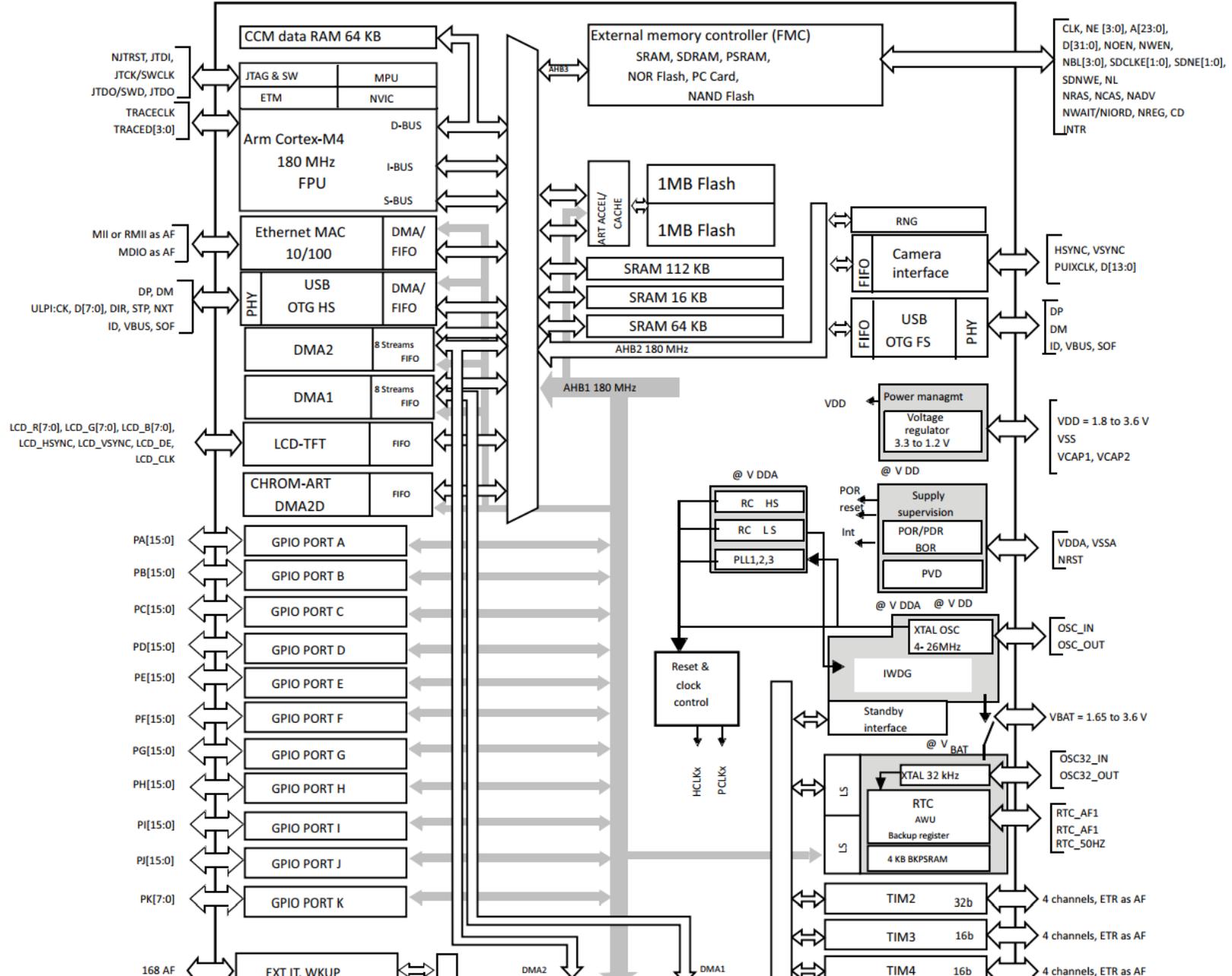
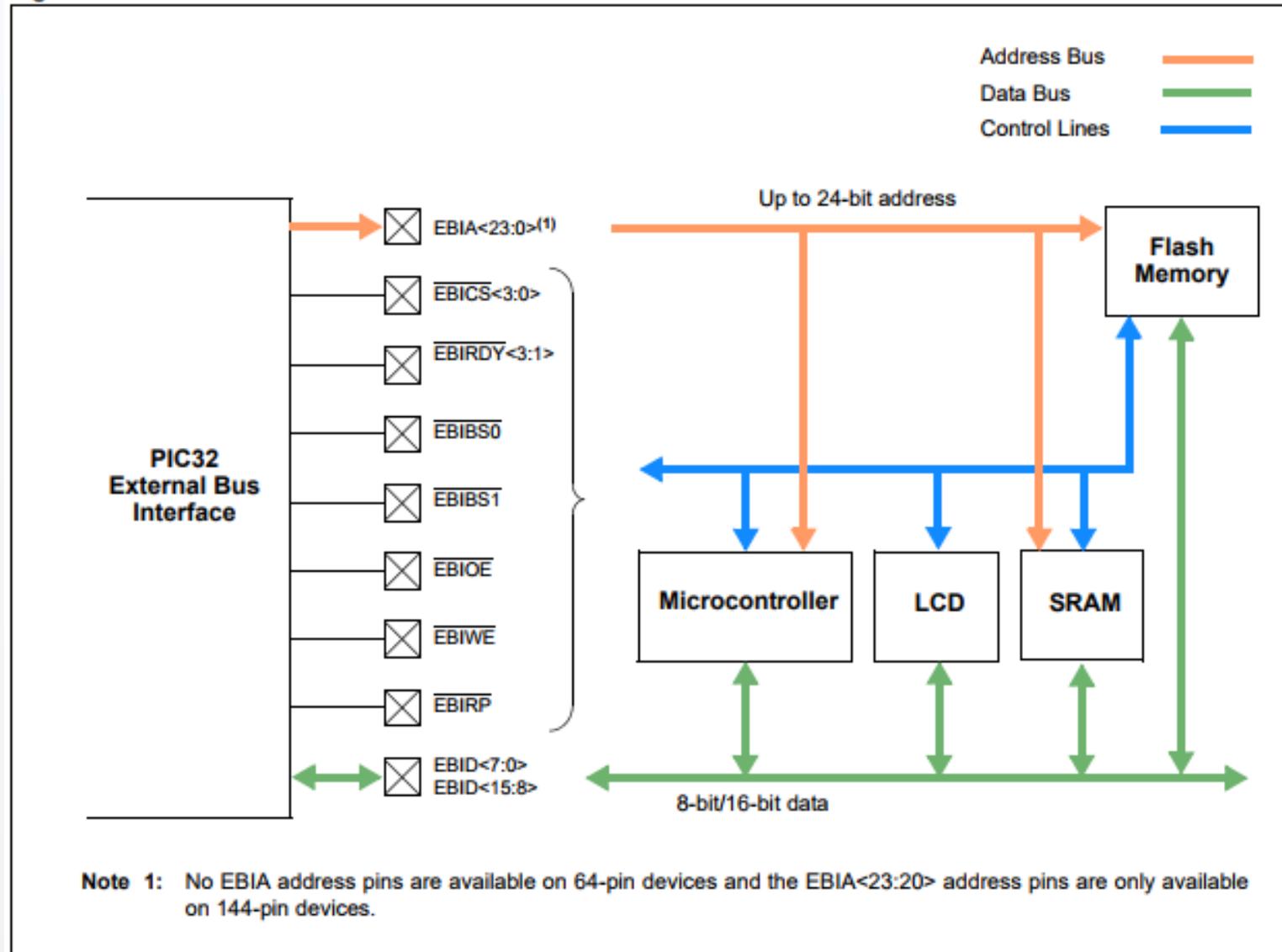
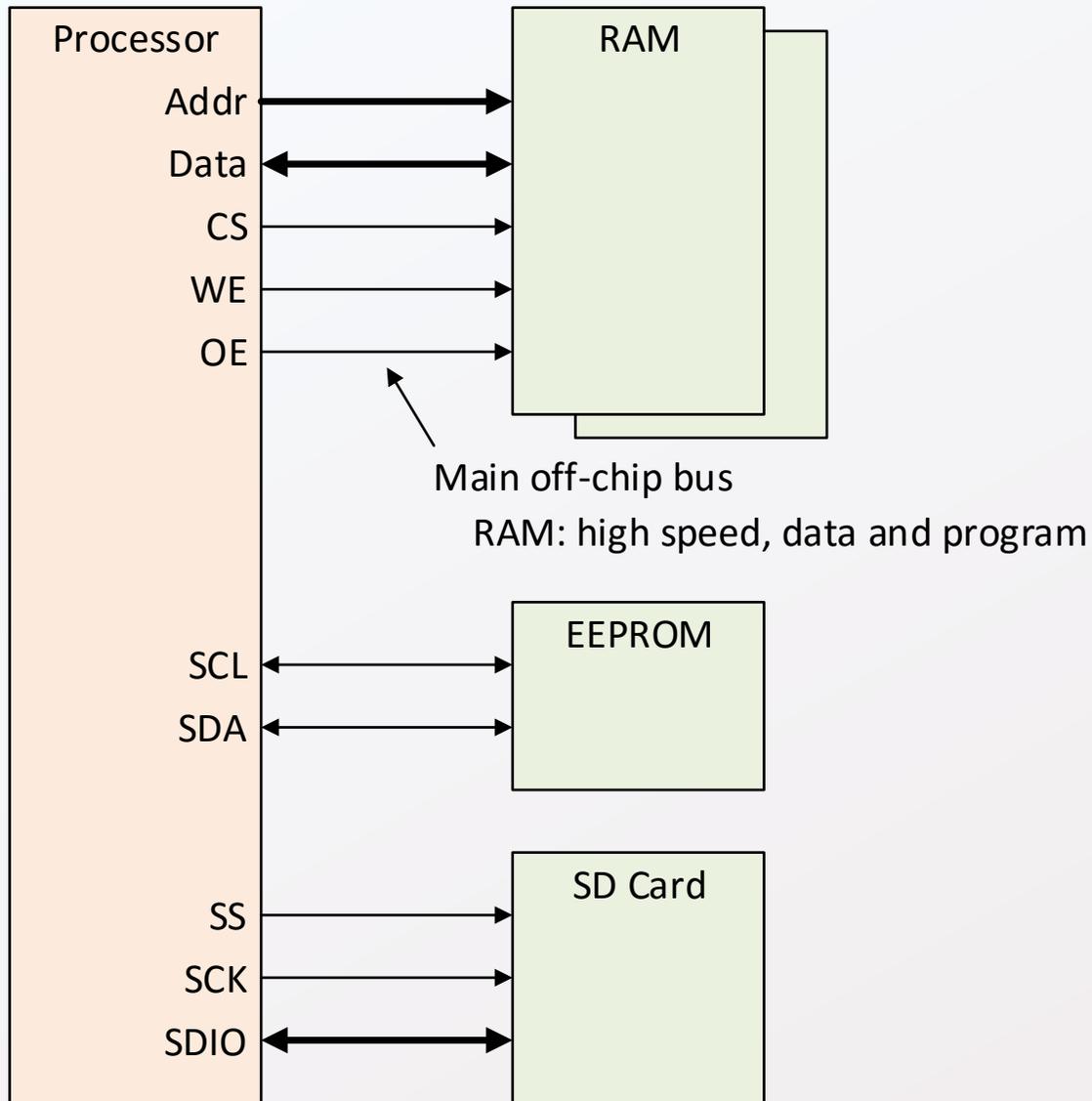


Figure 47-2: EBI Module Pinout and Connections to External Devices



A typical higher-performance configuration



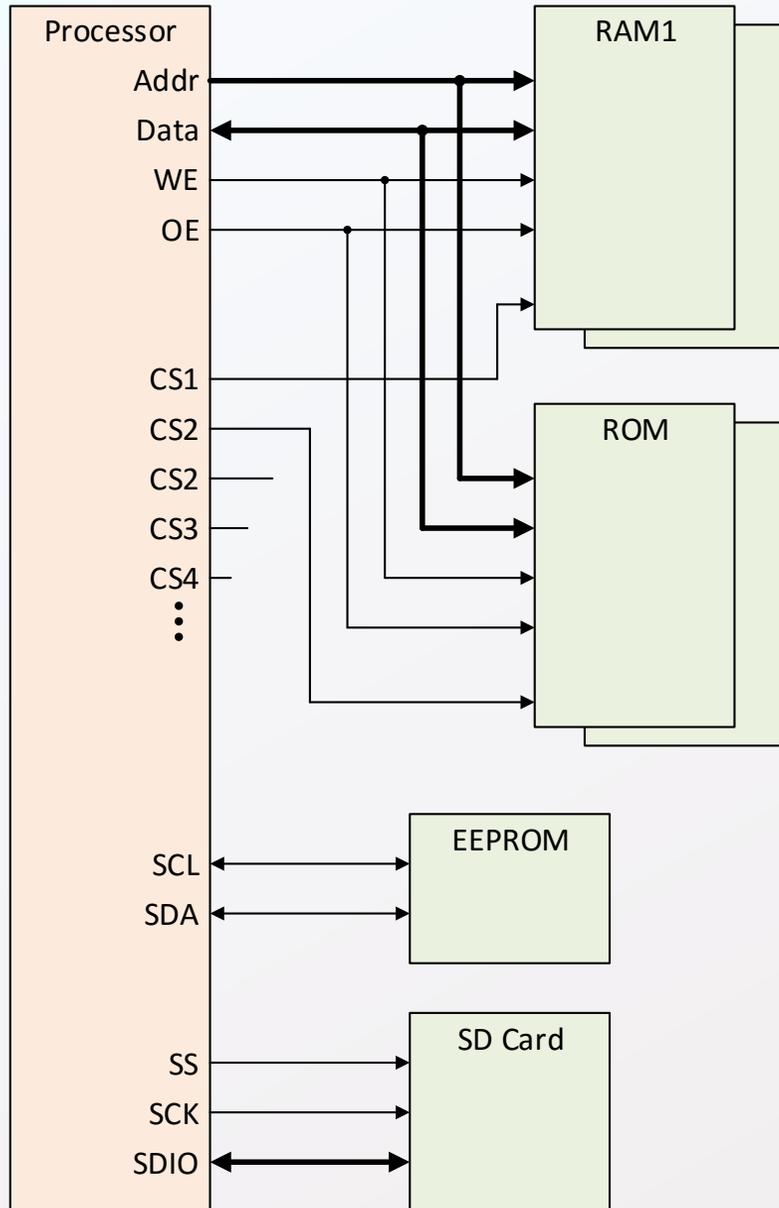
Single bank of homogenous RAM

If multiple chips, all are similar

System software loaded from ROM into RAM for execution

ROM may be HD/SSD, SD, or network

A typical lower-to-midrange-performance configuration



Single bank of homogenous RAM

If multiple chips, all are similar

Single bank of parallel ROM that shares same bus; timings similar enough

Booting

Ram is loaded from external source (like SD card) during “boot” sequence

(Some processors execute from parallel ROM)

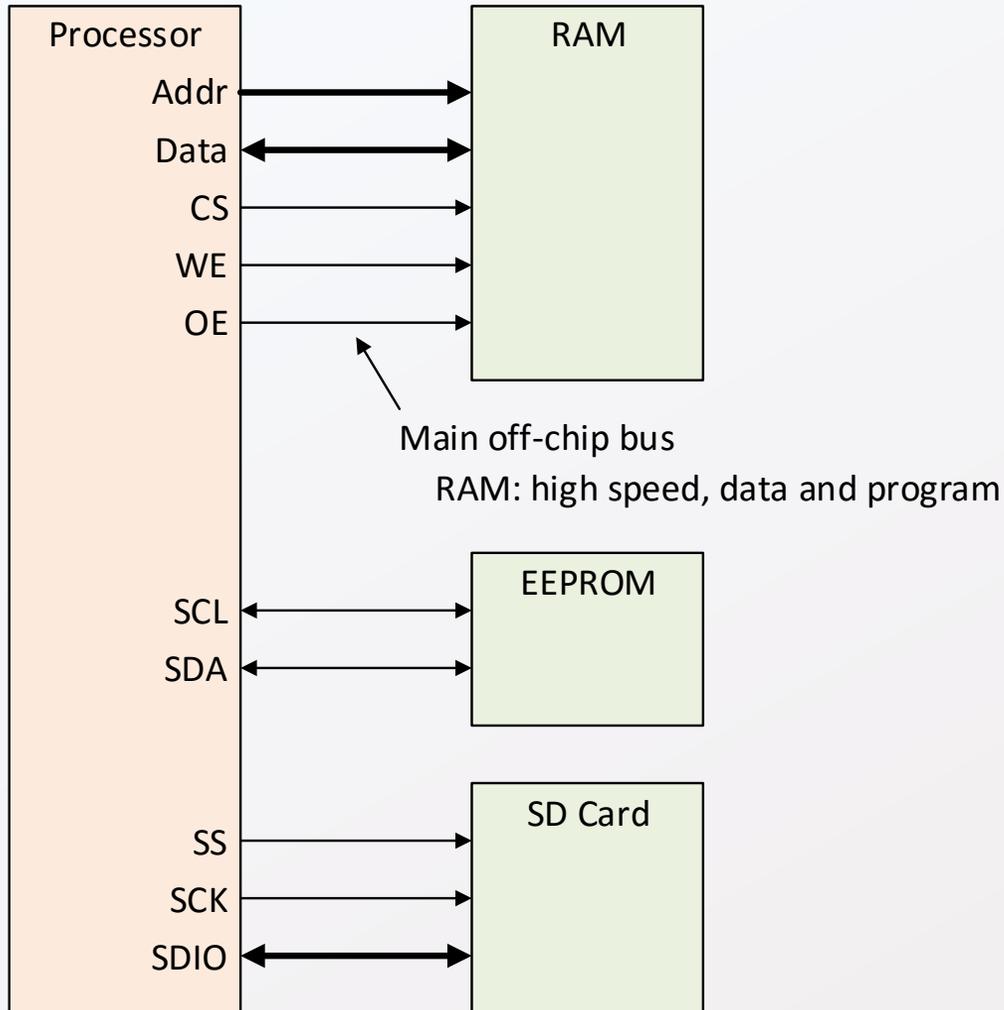
What is a “boot” sequence?

Pull oneself up by your own bootstraps – i.e., improve the situation without outside help, in small measured steps

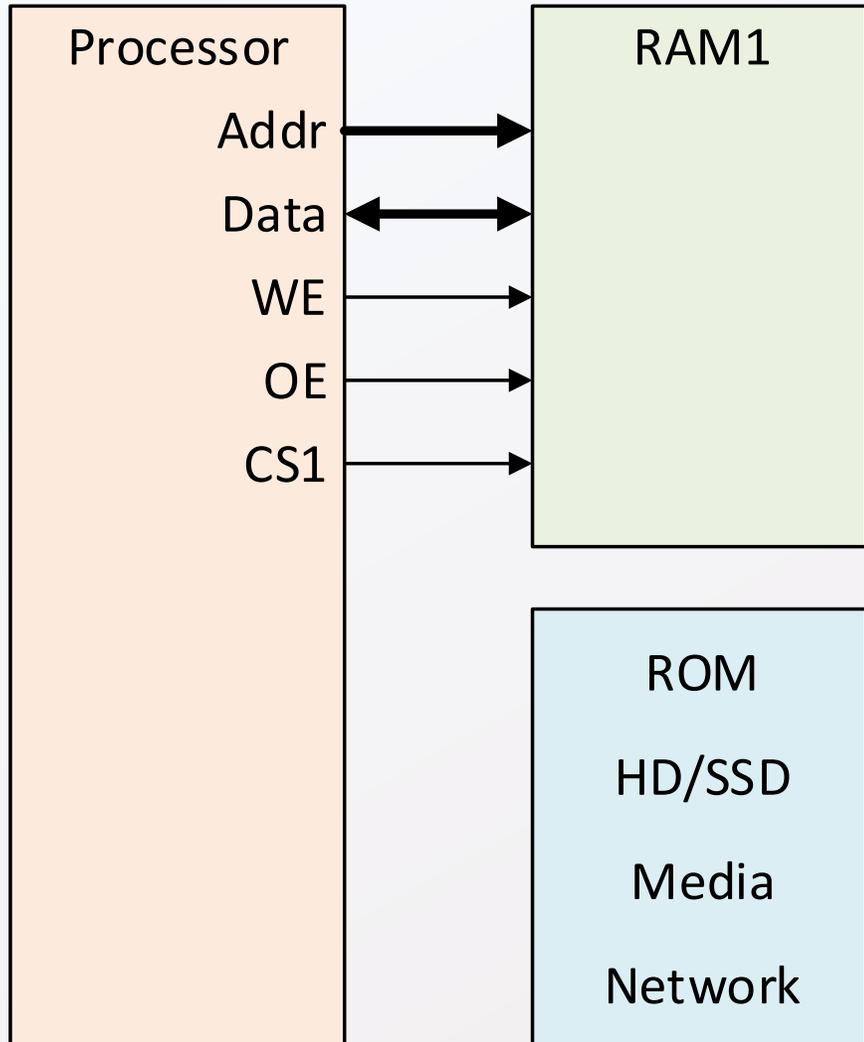
Power applied, reset released, then what?

Fetch first instruction... from where?

Fixed address; must contain ROM that responds to “natural bus cycle”



Program Execution



Source code you develop is stored in a file managed by the operating system. The tools compile the source, create an .elf (or other format) object file that can be placed in memory and directly executed.

The .elf was built using a symbol table that contains relative addresses for data and program redirects. It can be loaded anywhere in RAM for execution.

Where should it be placed? How can it be placed?

Should the source file creator need to know where it might end up?

Models

One programmer, one application: program can be “hard located” in memory

One programmer, multiple applications: programs may be “hard located” in memory

Multiple programmers: programs can't be “hard located” in memory – how could they be, given changing code size, changing stack/heap size, changing data needs

How to inform programmers where to locate programs? Can't!

Solution: Every program gets the same view of memory and resources – every program can assume it is the only program.

Implementation: Does the OS sort out what's loaded where? Would that even work?

Absolute vs. relative references in code base

Virtual Memory

Only practicable solution: hardware does dynamic address translation in real-time

Every single LOAD/STORE memory address passes through MMU and gets translated

All programs can be written as if they were the only program running

OS assigns physical memory location

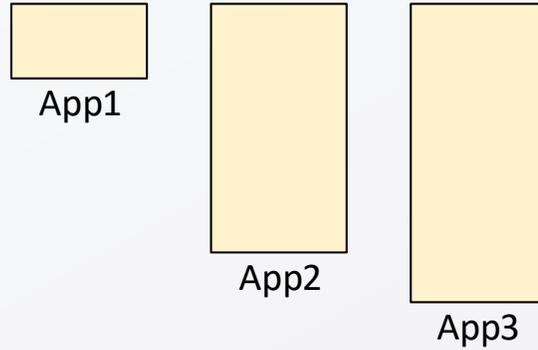
MMU maps every address to a physical address on the fly

Compiler/Assembler do not need to know about execution environment

Note: PC-relative indexing works regardless of physical addresses

Virtual Memory

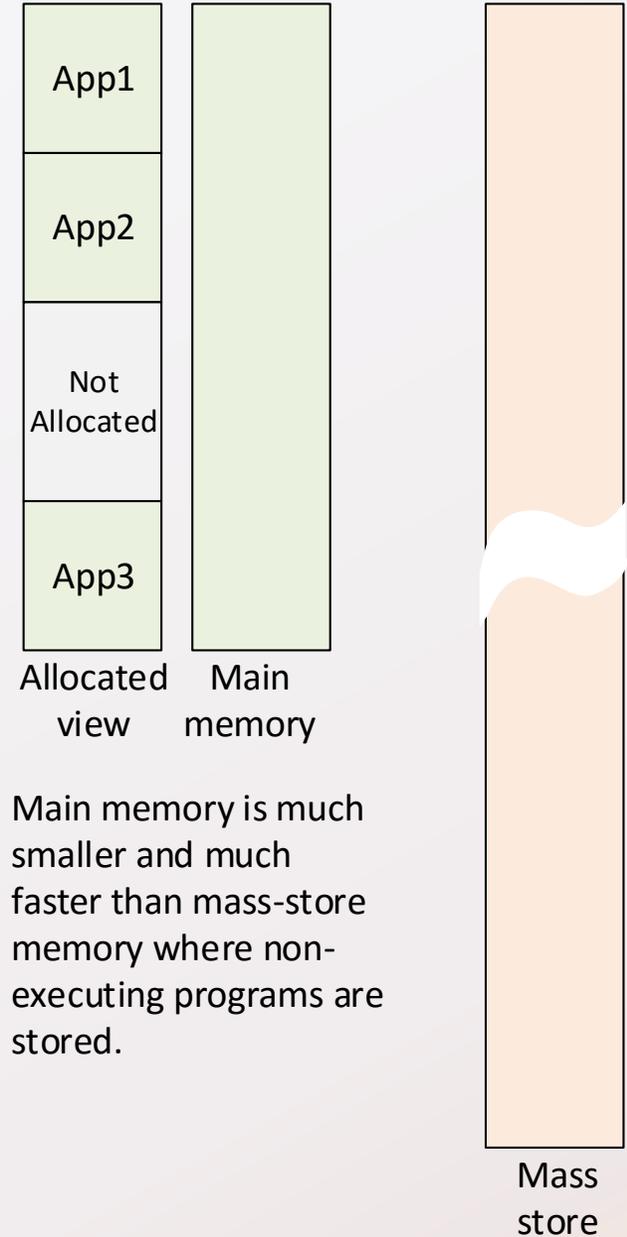
OS allocates memory to applications. Allocated memory need not be same size as executable code.



All application programs assume they start at address 0, and they rely on the OS to assign a location in physical memory, and the MMU to translate virtual addresses in real time.

Some applications are larger than their allocated memory.

More applications may be pending; some applications may need to surrender their main memory allocations to allow other programs to run.



Main memory is much smaller and much faster than mass-store memory where non-executing programs are stored.

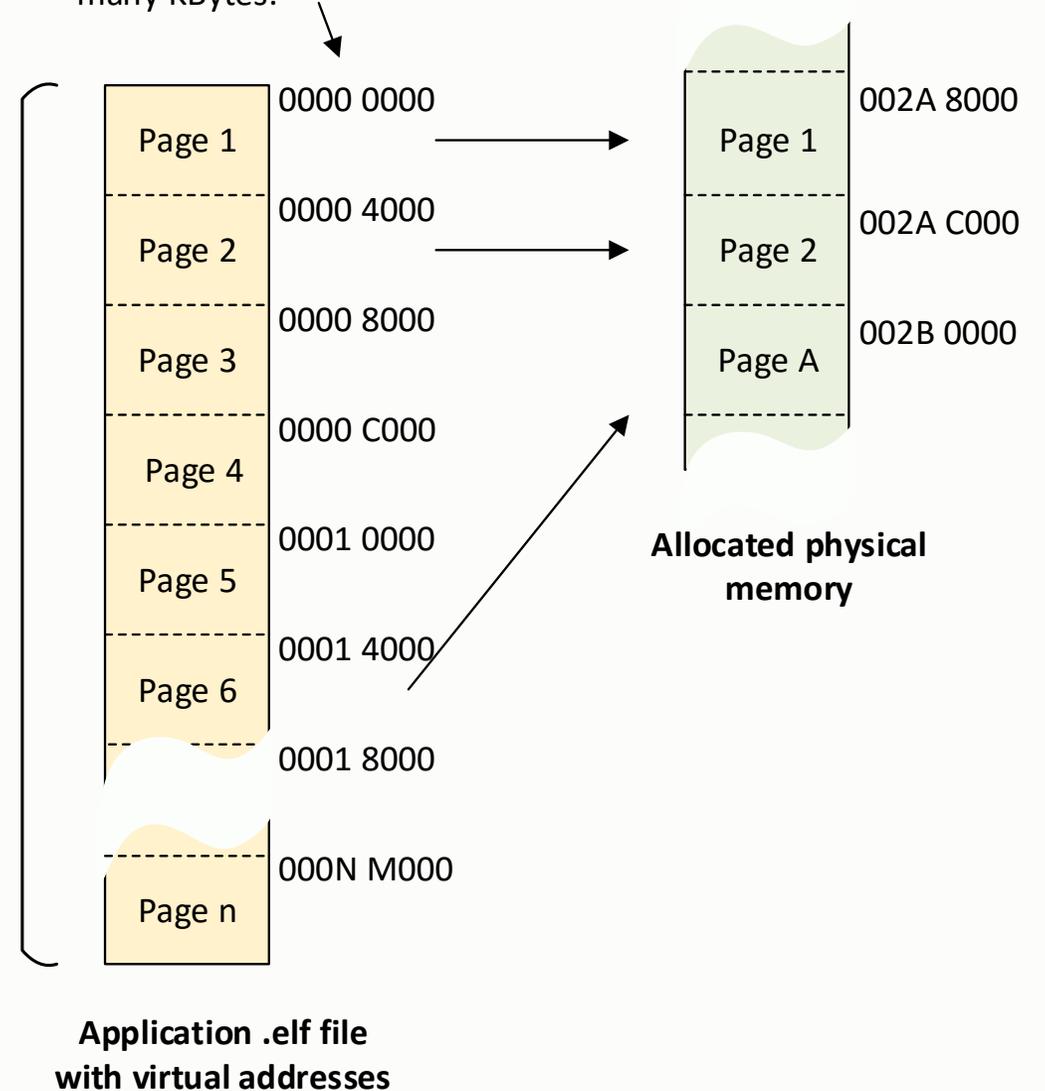
Virtual Memory

Files are divided into “pages”. Pages are swapped in and out of physical memory as needed.

If an access to a non-resident page is requested, a “page fault” occurs and processing must stop until page is loaded.

Application divided into “pages”; each page can be placed in main memory as needed

Virtual addresses showing example page sizes – pages range from a few KBytes to many KBytes.



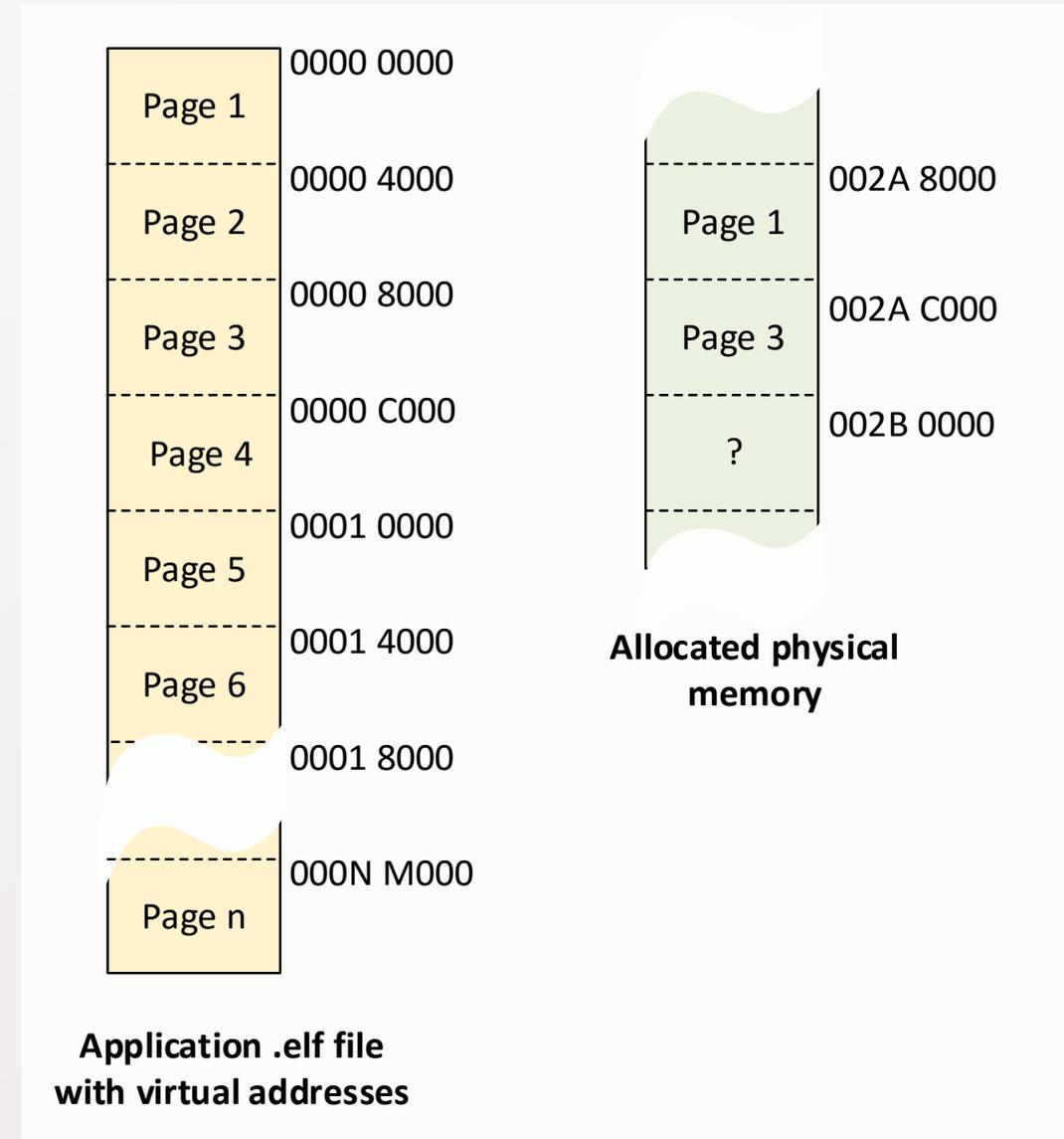
Virtual Memory: Constructing physical addresses

This example uses a 16K page; the lower 14 bits are the same for the virtual and physical address.

For a 32-bit address, the upper 18 bits must be “mapped” from the virtual address into the correct physical address space.

A “page table” stored in main memory stores the upper 18 bits of the physical base address of each page. The address stored for page 3 is 002AC.

Virtual address 0000 8123 would map to physical address 002A C123.



Virtual Memory: Translation look-aside buffer

If every translation required an access to main memory to retrieve the upper 16 bits, the memory would run at half speed. Some number of page table entries are stored in a cache called the TLB.

If the referenced page is in physical memory, it's "tag" (the upper 16 bits) will be in the TLB and the physical address can be constructed quickly.

How is the tag located? CAM.

