G54MDP
Mobile Device Programming

Mobile Phone Architecture

Brainstorming

- Pick a mobile device (perhaps one that you own)
- Think about what the characteristics and specifications of that device are
  - CPU speed, display type, size, shape, network connectivity, battery life etc
  - Operating system, mode of interaction / UI
- How might these affect how we program for it?

Learning Outcomes

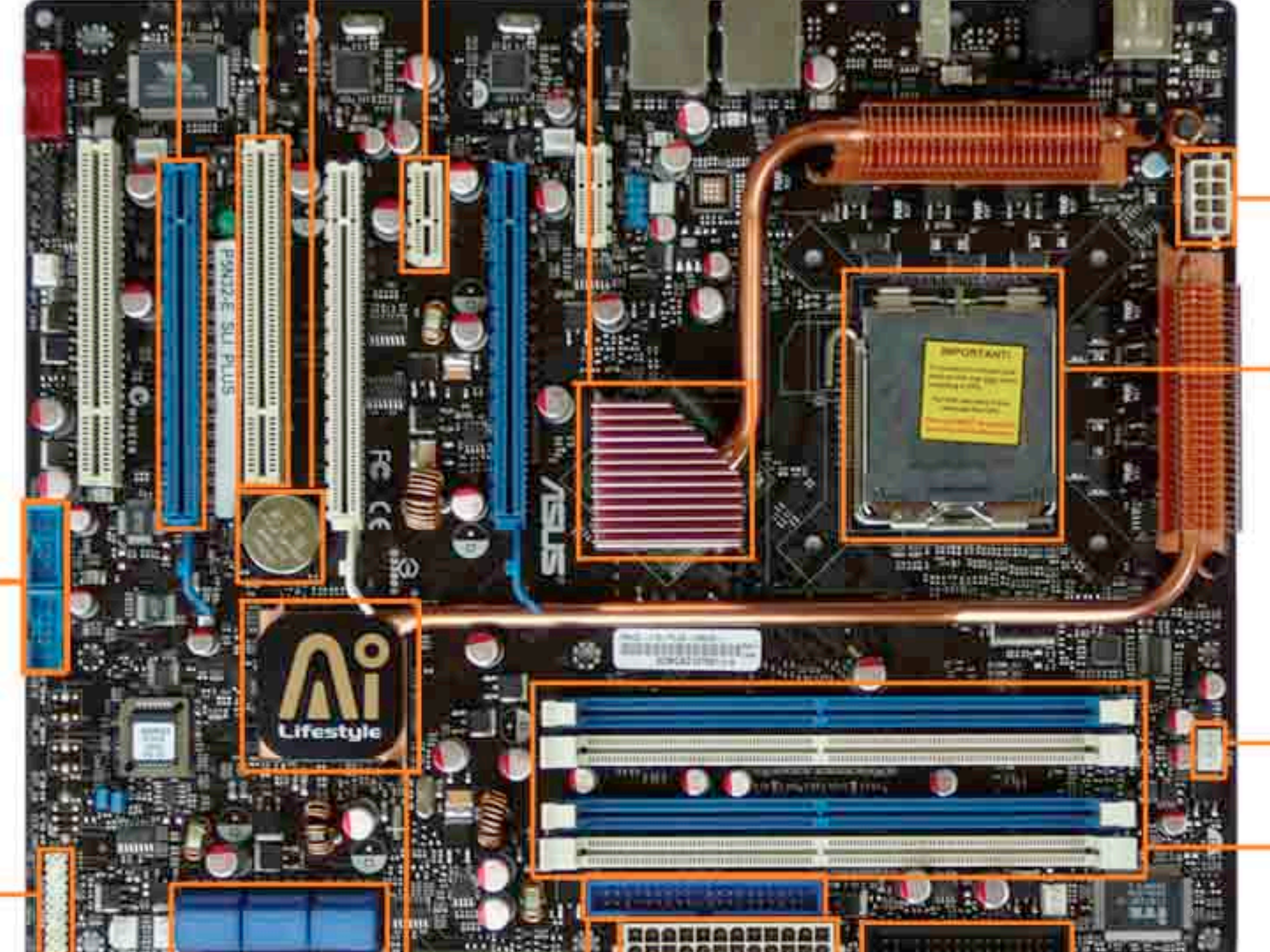
- By the end of this lecture you should…
- Understand the architectural differences between a PC and a mobile phone
- Have an overview of the main components of a mobile phone
- Understand some of the differences between ARM and x86 as relevant to mobile phones

Mobile Device Characteristics

- CPU ~1Gz
- GPU
- Memory
  - RAM 128MB-1GB
  - Flash Storage 16-64GB, internal / external via SD card
- Communications
  - Telephony
  - WiFi
  - Bluetooth
  - NFC
- Screen
- Audio
- User input
- Battery

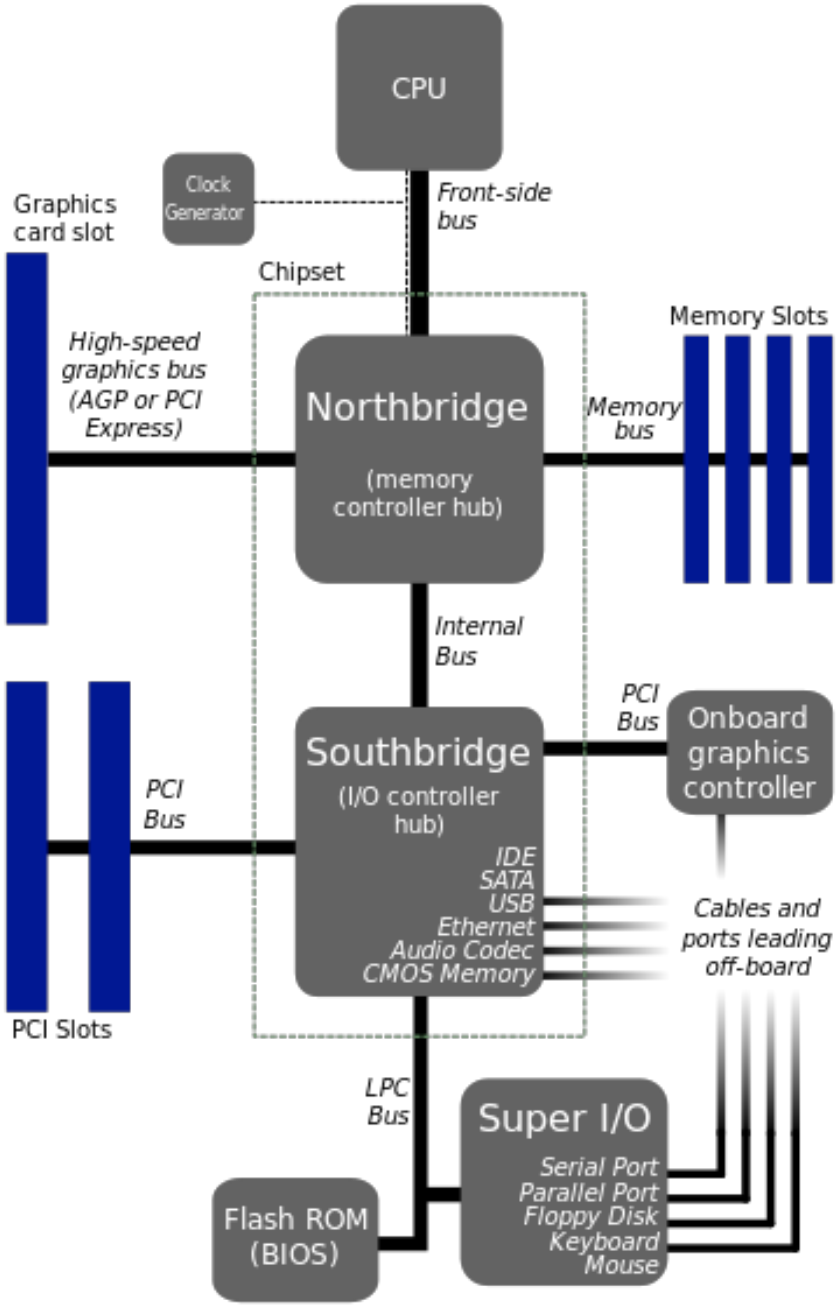
Sound familiar?

- In many ways, the technology inside a modern smart phone is conceptually very similar to that within a desktop PC
- It is the way that it is put together that is different


P5N32-E SLI PLUS
ASUS
IMPORTANT!
Ai
Lifestyle

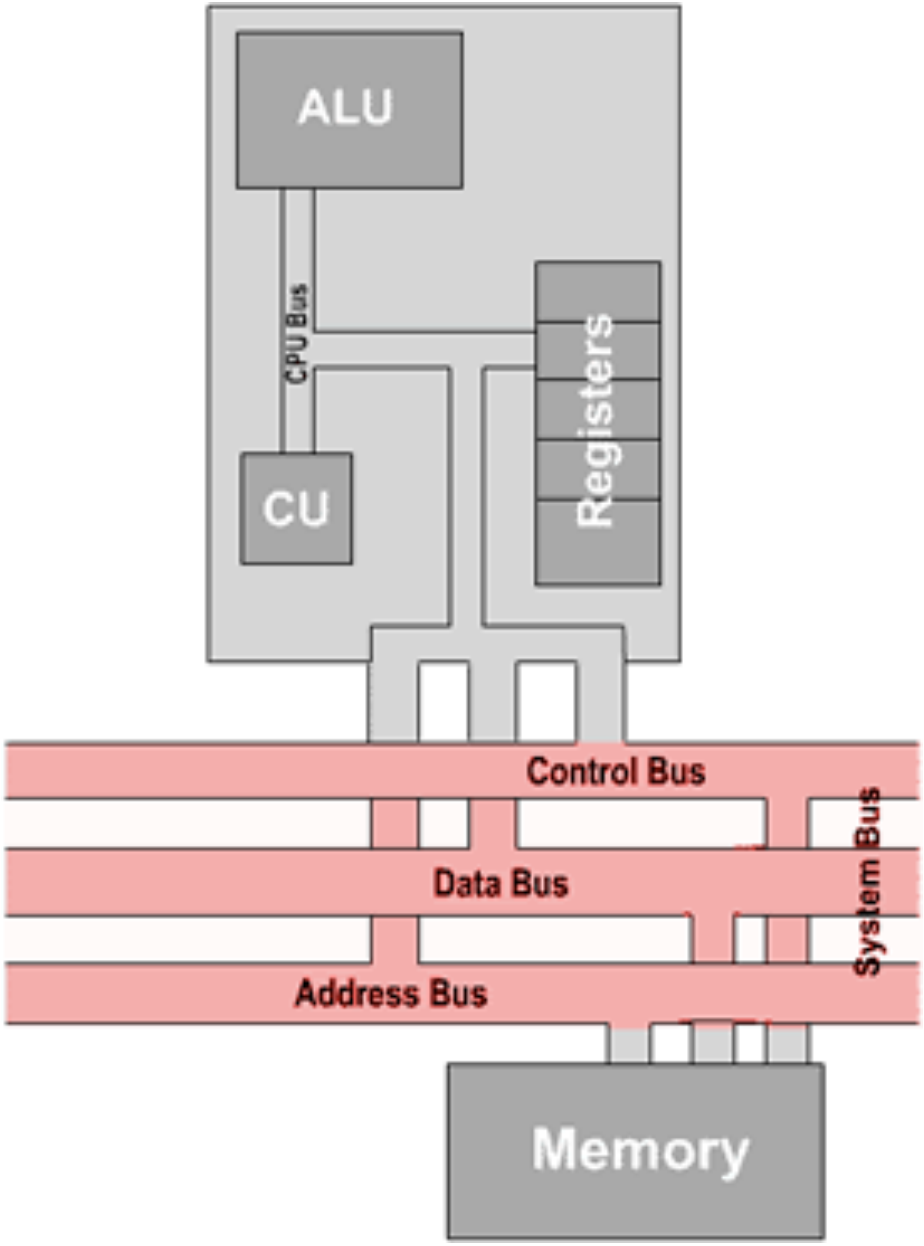
Inside a PC

- Several chips
- CPU
- Northbridge
- Southbridge
- RAM
- Several cards with extra bits
  - NIC, GPU, Audio
- Disks
  - HDD / SSD


CPU
Clock Generator
Front-side bus
Graphics card slot
Chipset
Memory Slots
High-speed graphics bus (AGP or PCI Express)
Northbridge
(memory controller hub)
Memory bus
Internal Bus
PCI Bus
Onboard graphics controller
Southbridge
(I/O controller hub)
PCI Bus
IDE
SATA
USB
Ethernet
Audio Codec
CMOS Memory
Cables and ports leading off-board
PCI Slots
LPC Bus
Super I/O
Serial Port
Parallel Port
Floppy Disk
Keyboard
Mouse
Flash ROM (BIOS)

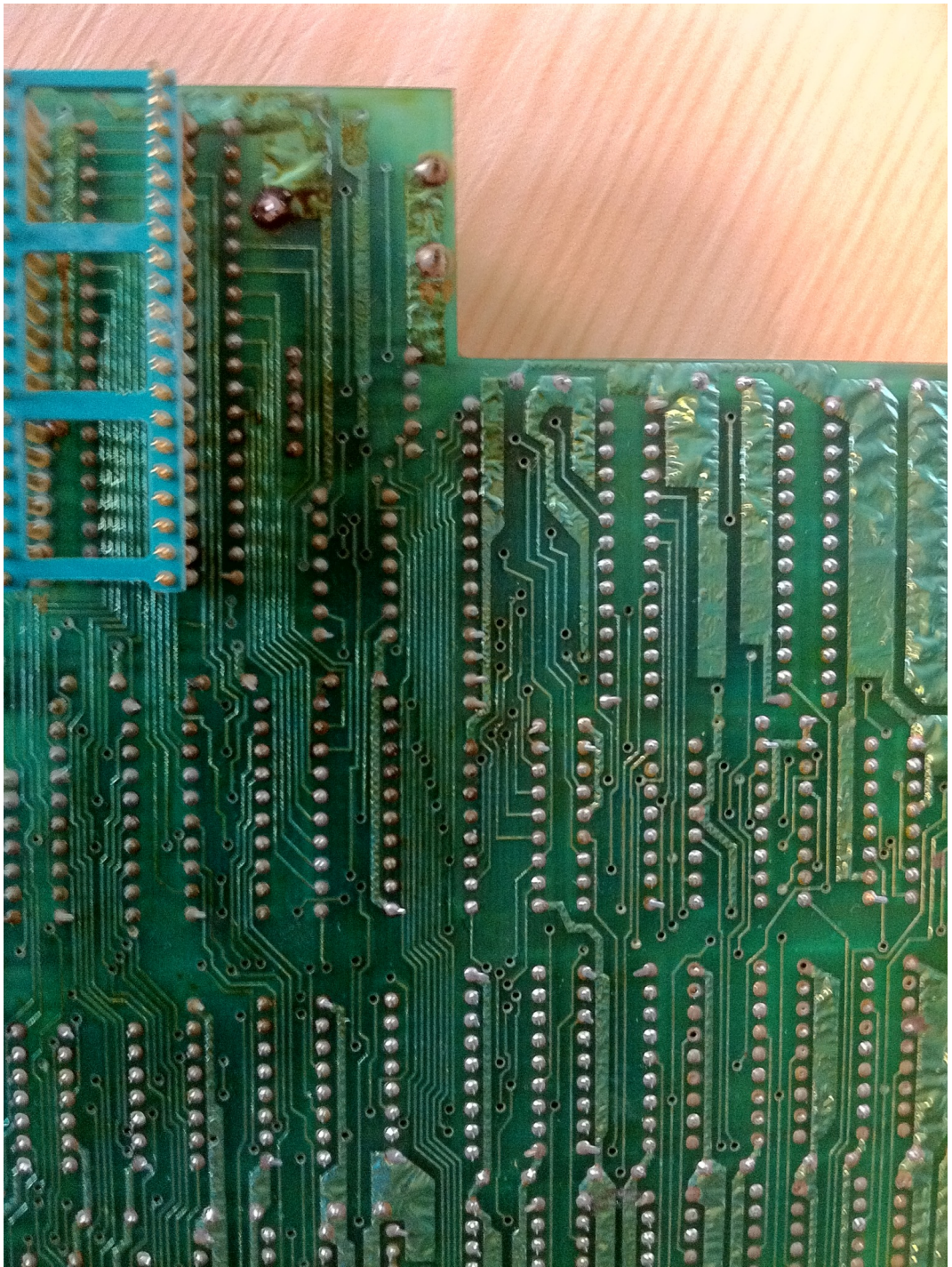
Connections

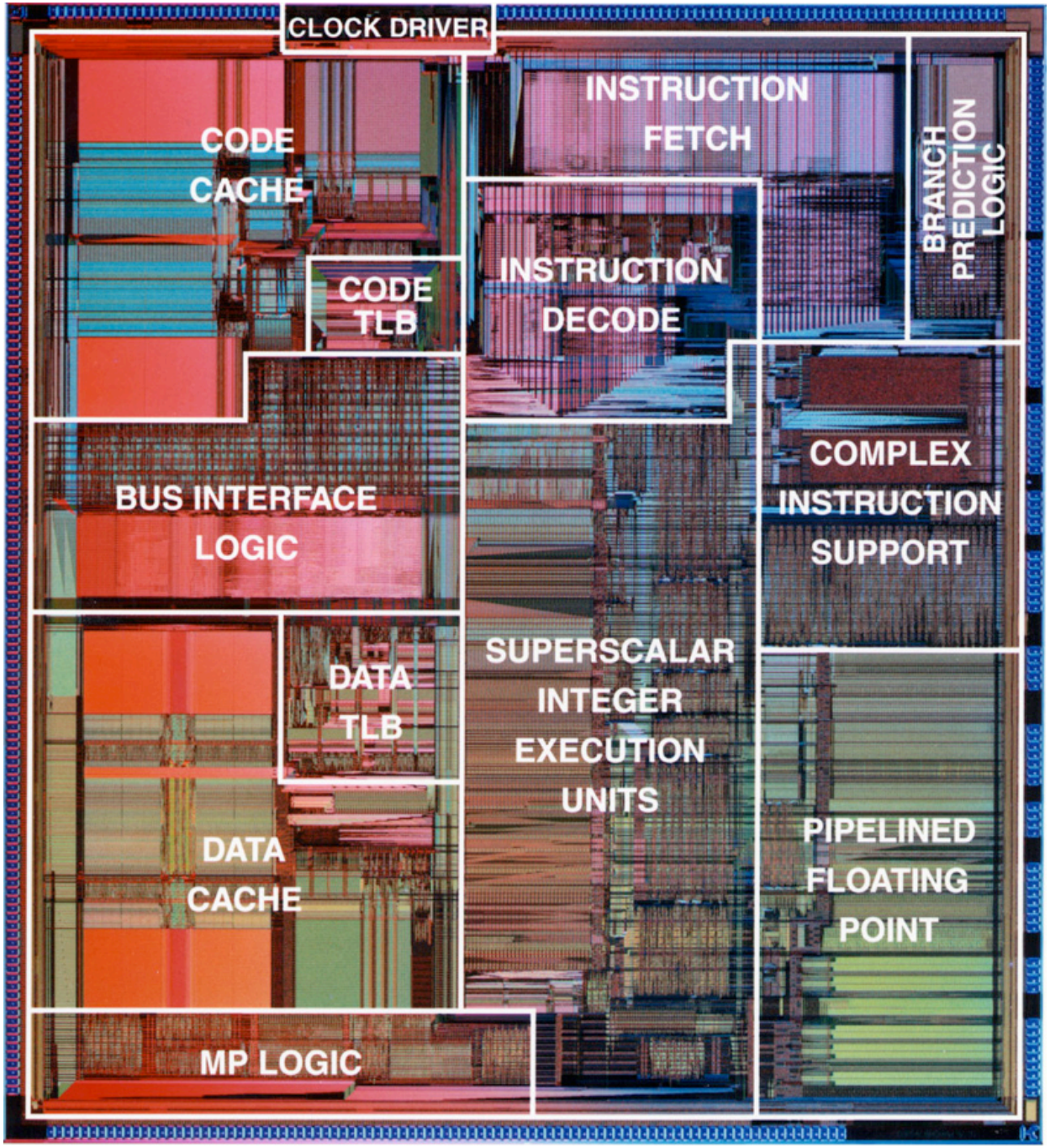
- CPU has a bus that connects it to the other devices / chips
- Originally, this would have been an address bus and a data bus, alongside some control buses
- RAM chips etc have similar connections
- Connect CPU to RAM / ROM
- Add some control logic and you have a PC


ALU
CPU Bus
Registers
CU
Control Bus
Data Bus
Address Bus
System Bus
Memory

Connections

- Nowadays more advanced buses are used
  - E.g. HyperBus
- Bus width
  - The more lines (wires) the more data can be transferred ~= faster bus
  - However, the number of pins used for the bus defines chip size
- Multiple chips and connections
  - Communications overhead / voltage drop = power
  - Clean architectural principle
  - Difficult to fit into a mobile device
  - Requires a different approach




CLOCK DRIVER
CODE CACHE
INSTRUCTION FETCH
BRANCH PREDICTION LOGIC
CODE TLB
INSTRUCTION DECODE
BUS INTERFACE LOGIC
COMPLEX INSTRUCTION SUPPORT
DATA TLB
SUPERSCALAR INTEGER EXECUTION UNITS
DATA CACHE
PIPELINED FLOATING POINT
MP LOGIC

Transistors

- A CPU is a collection of transistors
- All digital logic devices are built out of transistors
- 4 transistors will build a NAND gate
- Any logic circuit can be built using multiple NAND gates
- Getting better at getting lots of transistors on a chip
  - Moores law

System on a Chip

- Only use some of the transistors on a chip to form the CPU
- Use the rest to build the other required components of the system
- External pins connect directly to peripheral hardware, not core chips
- Called System on a Chip (SoC)
  - The whole system is literally on a single chip
    - Integrates several heterogeneous components on a chip
  - Aims to reduce communications overhead

System on a Chip

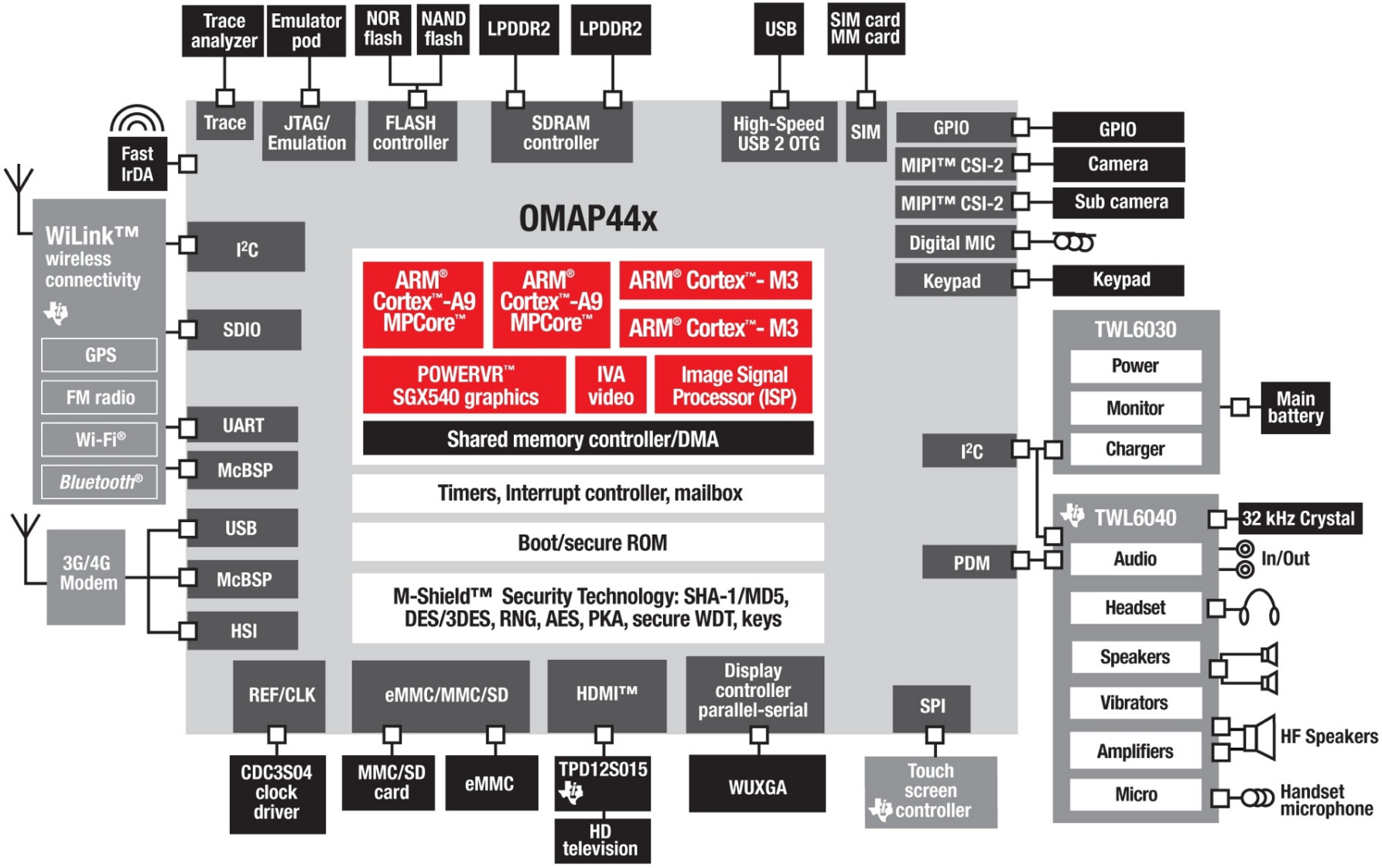
- Transistors provide computation, storage
  - Divide the chip into multiple communicating regions
- Built block by block from descriptions of separate parts
  - IP (intellectual property) cores
    - CPU core
    - GPU core
    - RAM core

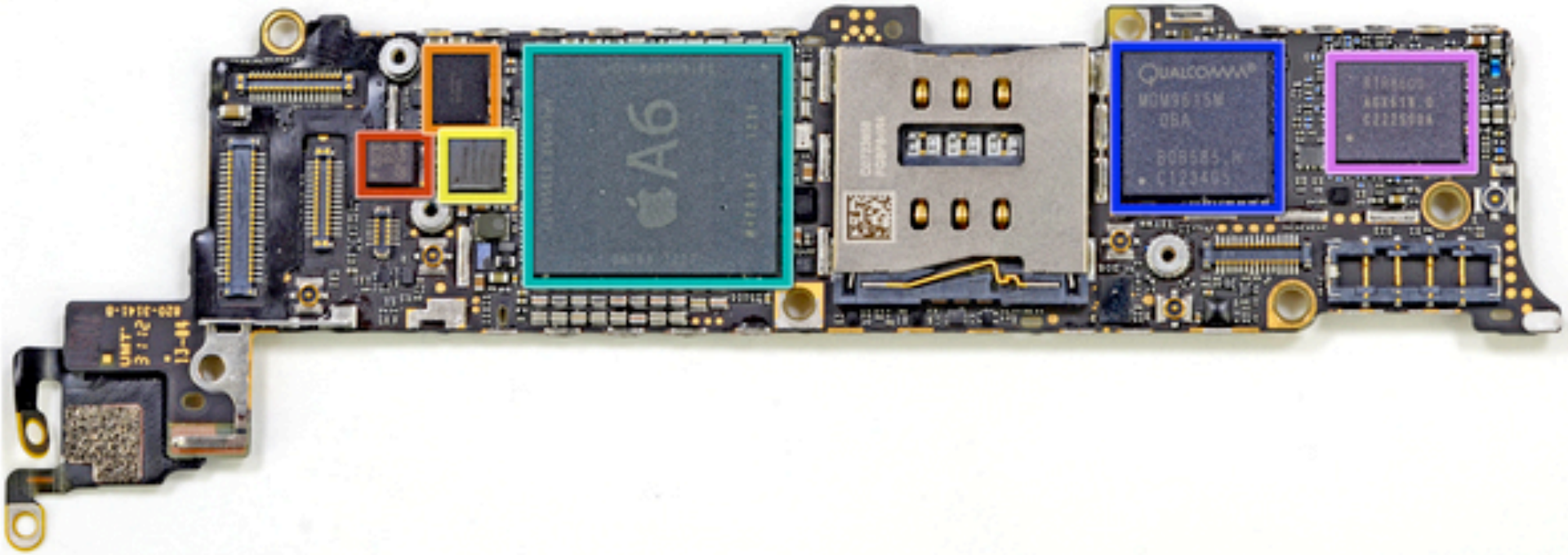
System on a Chip

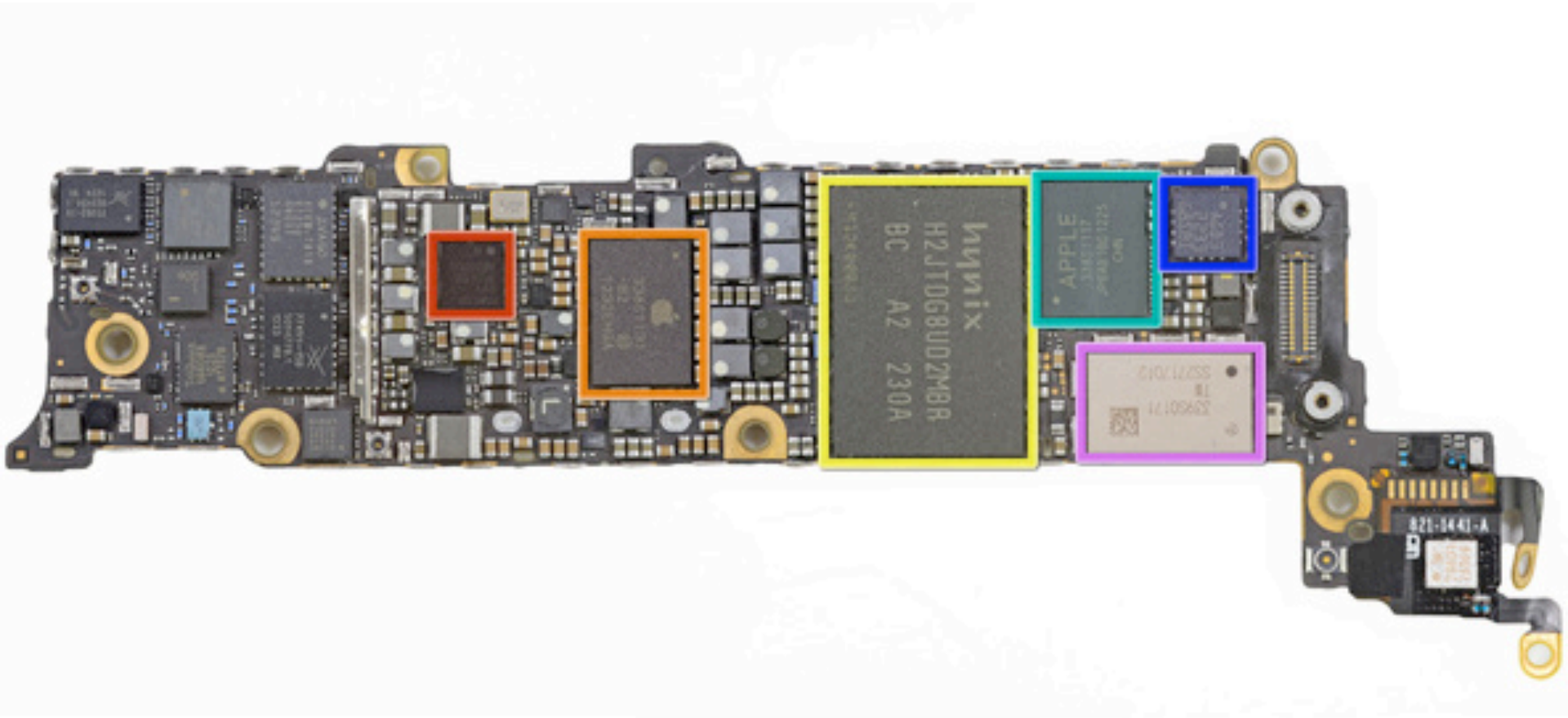
- Qualcomm Snapdragon
  - Nexus 4, Droid Razr, HTC Evo
- Apple A4/A5/A6/A7
  - iPhones
- Texas Instruments OMAP
  - Kindle Fire, Nook
- All share the same CPU block
  - ARM Cortex A8
- But may have different companion blocks

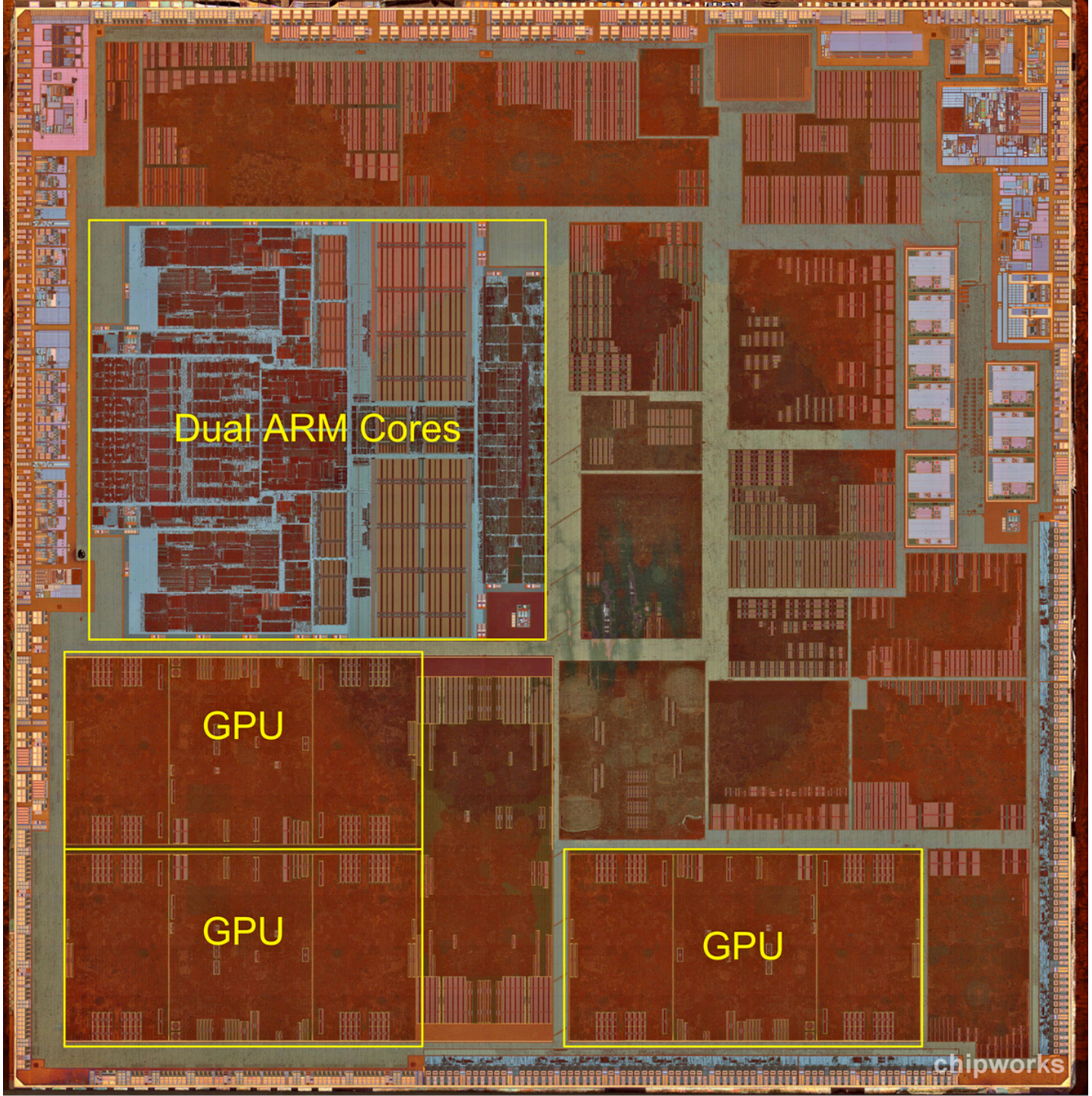
OMAP 44x0

- A typical SoC chip used in mobile phones
- Motorola Droid Bionic, Samsung Galaxy Nexus
- Contains two ARM Cortex-A9 CPU
- 3D GPU (PowerVR SGX)
- IVA Accelerator
- Image Signal Processor
- Some analogue components have a limited minimum size
  - Analogue vs digital – why?


Trace analyzer
Emulator pod
NOR flash
NAND flash
LPDDR2
LPDDR2
USB
SIM card MM card
Trace
JTAG/ Emulation
FLASH controller
SDRAM controller
High-Speed USB 2 OTG
SIM
GPIO
MIPI™ CSI-2
MIPI™ CSI-2
Digital MIC
Keypad
GPIO
Camera
Sub camera
Keypad
Fast IrDA
WiLink™ wireless connectivity
GPS
FM radio
Wi-Fi®
Bluetooth®
I²C
SDIO
UART
McBSP
OMAP44x
ARM® Cortex™-A9 MPCore™
ARM® Cortex™-A9 MPCore™
ARM® Cortex™- M3
ARM® Cortex™- M3
POWERVR™ SGX540 graphics
IVA video
Image Signal Processor (ISP)
Shared memory controller/DMA
Timers, Interrupt controller, mailbox
Boot/secure ROM
M-Shield™ Security Technology: SHA-1/MD5, DES/3DES, RNG, AES, PKA, secure WDT, keys
TWL6030
Power
Monitor
Charger
Main battery
I²C
PDM
TWL6040
32 kHz Crystal
Audio
In/Out
Headset
Speakers
Vibrators
Amplifiers
Micro
HF Speakers
Handset microphone
3G/4G Modem
USB
McBSP
HSI
REF/CLK
eMMC/MMC/SD
HDMI™
Display controller parallel-serial
SPI
CDC3S04 clock driver
MMC/SD card
eMMC
TPD12S015
HD television
WUXGA
Touch screen controller

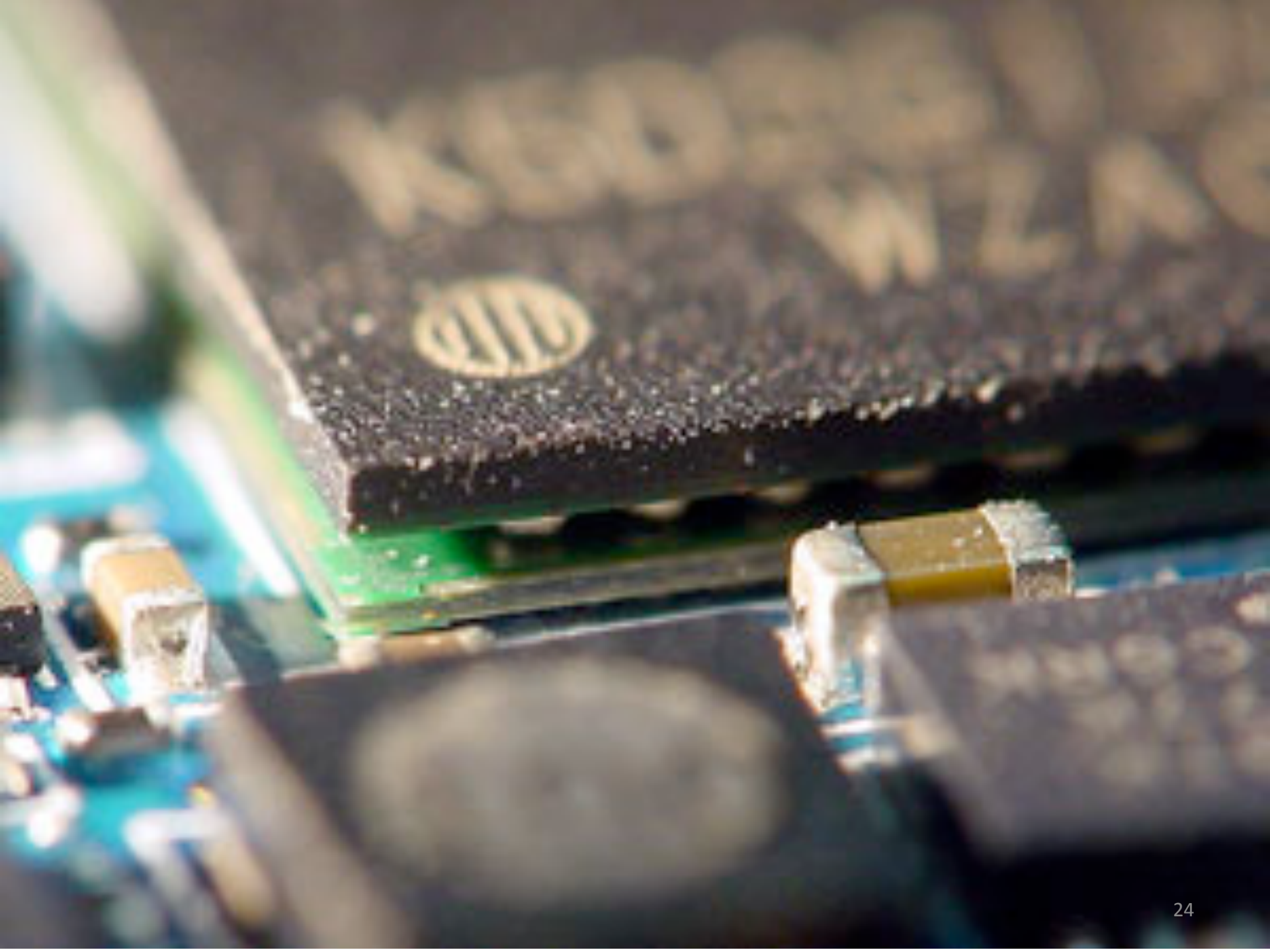

A6
QUALCOMM
MDM9615M
DBA

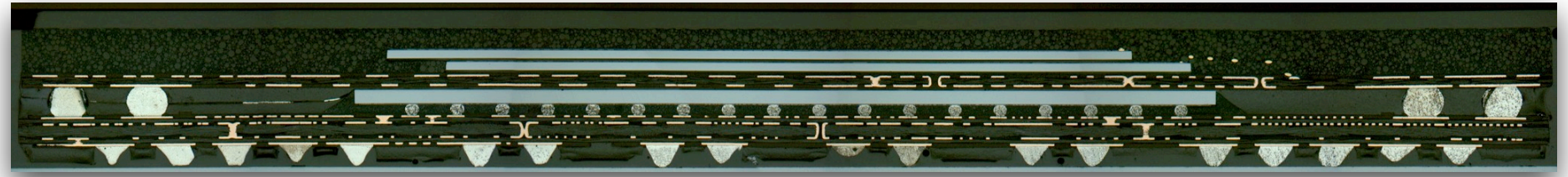

hynix
H2JTDG8UD2MBR
BC A2 230A
APPLE
821-3441-A


Dual ARM Cores
GPU
GPU
GPU
chipworks

Package-on-Package

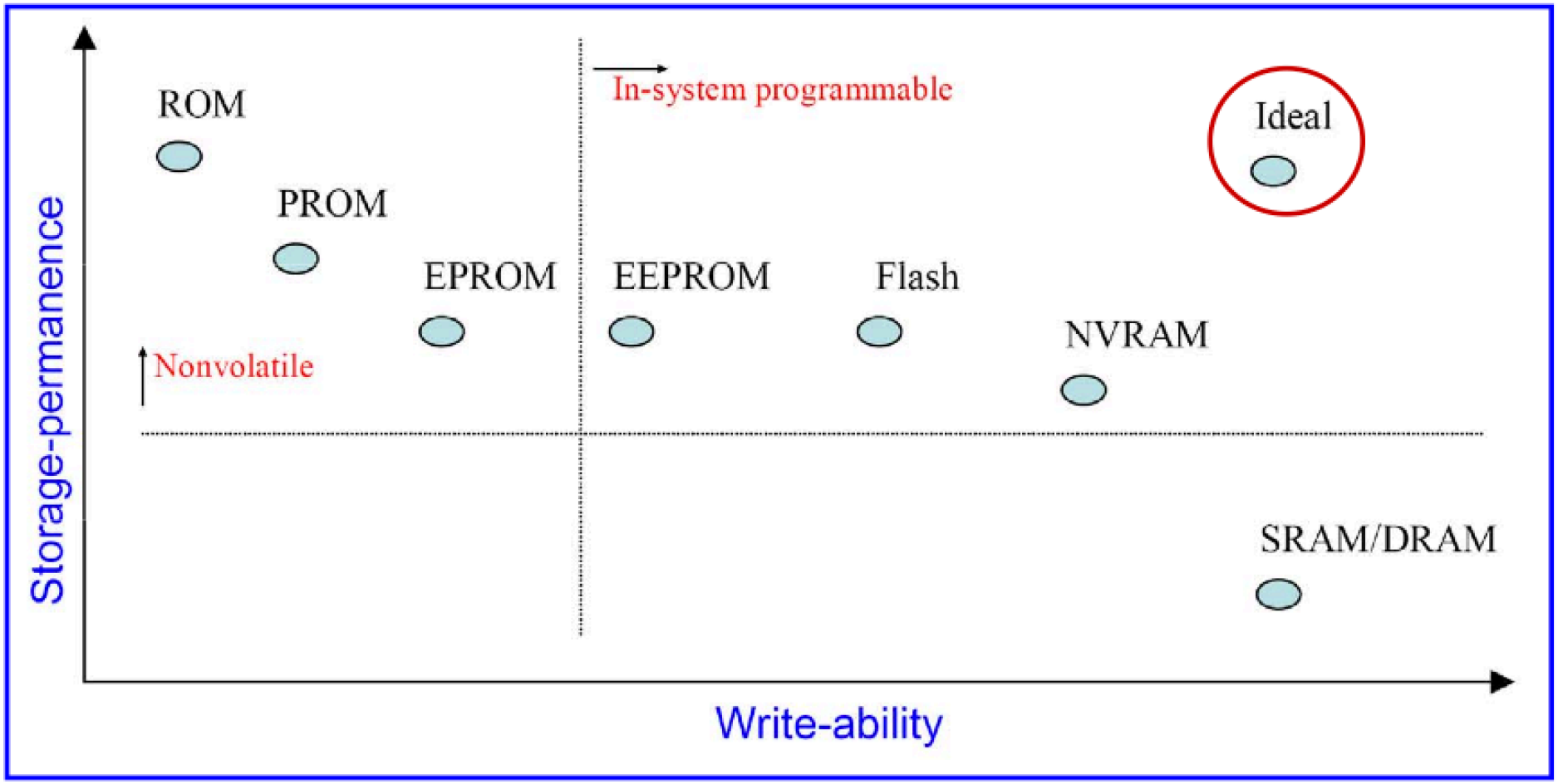
- What about RAM?
- Could add it to the SoC...
- But RAM uses a lot of space...
- Space we want to use for ‘useful’ stuff
- Separate package on top of the SoC
- Package-on-Package (or PoP)
  - Separate memory and logic production
  - “Stack” packages to minimise space





RAM

- RAM in a mobile device tends to be shared between multiple components on the main chip
  - C.f. laptop onboard GPU
  - Unlike separate motherboard RAM / GPU RAM
- Not all the RAM is available to the OS as “CPU RAM”
- No swap partition / page file
  - Why not?
  - What happens when you allocate all of the memory?
- Code assuming you don’t have very much


ROM
PROM
EPROM
EEPROM
Flash
NVRAM
SRAM/DRAM
Ideal
In-system programmable
Nonvolatile
Storage-permanence
Write-ability

Mobile CPUs

- Almost all Mobile Devices use ARM CPUs
- Originated from Acorn computers in the late-1980s
- Acorn RISC Machine
- Spun out from Acorn in early-1990s
- Now, Advanced RISC machine

ARM CPU

- 32-bit CPU (increasingly 64-bit)
  - Multiple instruction sets
  - Older phones made use of Jazelle DBX / embedded systems
    - Run Java bytecode directly, dedicated chips
- Sold as a design, not a physical device
  - Great for SoC usage
  - Chip manufactures can add ARM CPU to bespoke SoC vs buying a chip from Intel
- Aims
  - Fast and efficient
    - Good for mobile applications
    - Best use of battery – more instructions = more battery use
  - High code density
    - Less moving stuff around, best use of space

ARM CPU

- RISC-design (Reduced Instruction Set Chip)
- Only includes simple instructions that can be executed in one clock cycle
- Remove instructions you might expect in x86
  - e.g. divide instructions
  - If you need to divide, you roll your own in software
  - Or rather, the compiler does it for you

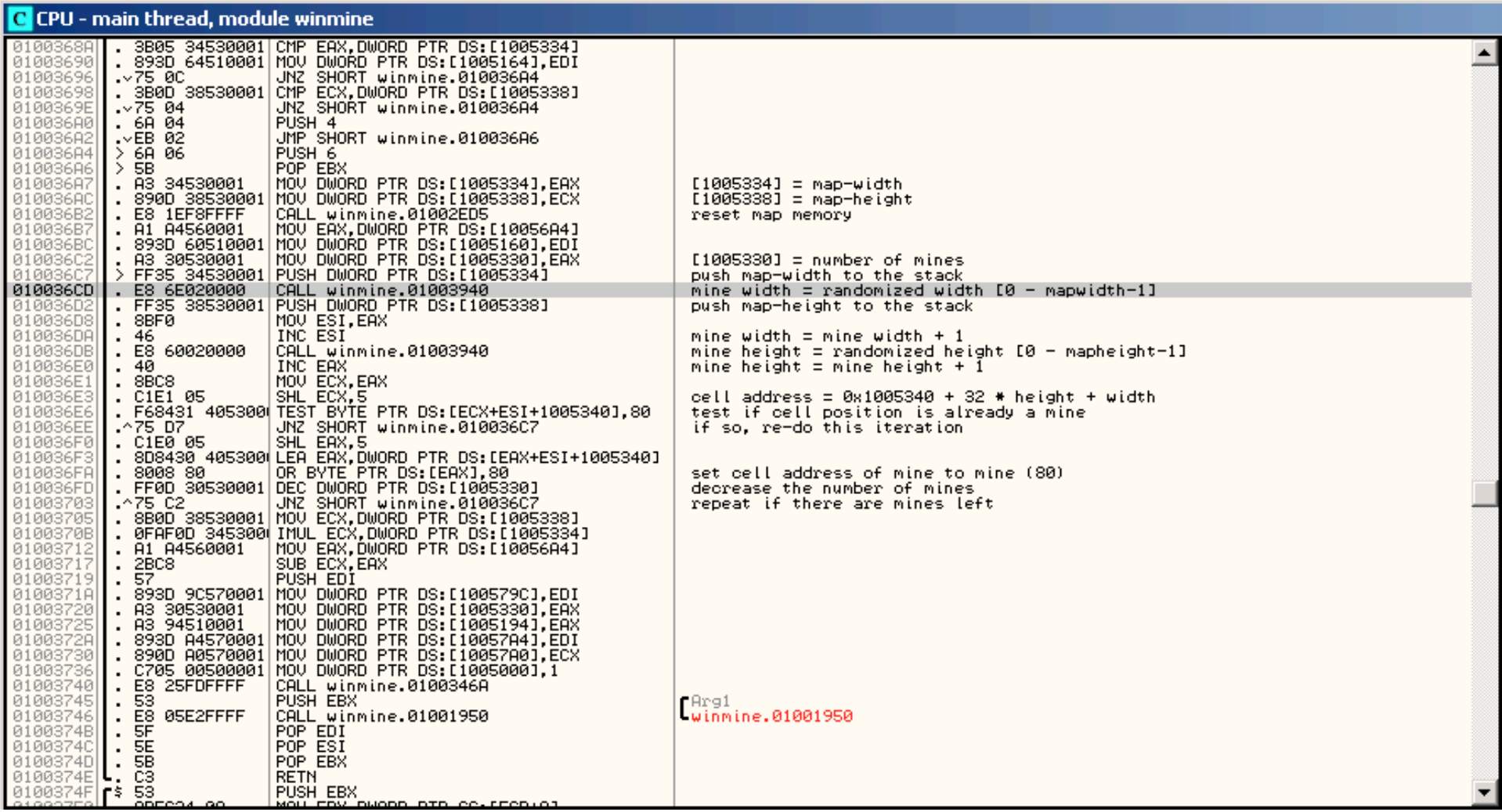
ARM CPU

Register names

r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	r14	r15

- 16 registers
- Load/Store architecture
- Each instruction is 32-bits long
- Makes decoding easy / efficient
  - Vs x86 variable instruction length
- But means loading a constant into a register can be tricky

Opcode	Type	Decoded Instruction
`0xE3A00000`	Data Processing	`MOV R0, #0`
`0xEF000002`	Software Interrupt	`SWI 2`
`0xEAFFFFFC`	Branch Instruction	`B -16`
`0xE0810002`	Data Processing	`ADD R0, R1,R2`


CPU - main thread, module winmine
0100368A . 3B05 34530001 CMP EAX,DWORD PTR DS:[1005334]
01003690 . 893D 64510001 MOV DWORD PTR DS:[1005164],EDI
01003696 .v75 0C JNZ SHORT winmine.010036A4
01003698 . 3B0D 38530001 CMP ECX,DWORD PTR DS:[1005338]
0100369E .v75 04 JNZ SHORT winmine.010036A4
010036A0 . 6A 04 PUSH 4
010036A2 .vEB 02 JMP SHORT winmine.010036A6
010036A4 > 6A 06 PUSH 6
010036A6 > 5B POP EBX
010036A7 . A3 34530001 MOV DWORD PTR DS:[1005334],EAX [1005334] = map-width
010036AC . 890D 38530001 MOV DWORD PTR DS:[1005338],ECX [1005338] = map-height
010036B2 . E8 1EF8FFFF CALL winmine.01002ED5 reset map memory
010036B7 . A1 A4560001 MOV EAX,DWORD PTR DS:[10056A4]
010036BC . 893D 60510001 MOV DWORD PTR DS:[1005160],EDI
010036C2 . A3 30530001 MOV DWORD PTR DS:[1005330],EAX [1005330] = number of mines
010036C7 > FF35 34530001 PUSH DWORD PTR DS:[1005334] push map-width to the stack
010036CD . E8 6E020000 CALL winmine.01003940 mine width = randomized width [0 - mapwidth-1]
010036D2 . FF35 38530001 PUSH DWORD PTR DS:[1005338] push map-height to the stack
010036D8 . 8BF0 MOV ESI,EAX
010036DA . 46 INC ESI mine width = mine width + 1
010036DB . E8 60020000 CALL winmine.01003940 mine height = randomized height [0 - mapheight-1]
010036E0 . 40 INC EAX mine height = mine height + 1
010036E1 . 8BC8 MOV ECX,EAX
010036E3 . C1E1 05 SHL ECX,5 cell address = 0x1005340 + 32 * height + width
010036E6 . F68431 405300 TEST BYTE PTR DS:[ECX+ESI+1005340],80 test if cell position is already a mine
010036EE .^75 D7 JNZ SHORT winmine.010036C7 if so, re-do this iteration
010036F0 . C1E0 05 SHL EAX,5
010036F3 . 8D8430 405300 LEA EAX,DWORD PTR DS:[EAX+ESI+1005340]
010036FA . 8008 80 OR BYTE PTR DS:[EAX],80 set cell address of mine to mine (80)
010036FD . FF0D 30530001 DEC DWORD PTR DS:[1005330] decrease the number of mines
01003703 .^75 C2 JNZ SHORT winmine.010036C7 repeat if there are mines left
01003705 . 8B0D 38530001 MOV ECX,DWORD PTR DS:[1005338]
0100370B . 0FAF0D 345300 IMUL ECX,DWORD PTR DS:[1005334]
01003712 . A1 A4560001 MOV EAX,DWORD PTR DS:[10056A4]
01003717 . 2BC8 SUB ECX,EAX
01003719 . 57 PUSH EDI
0100371A . 893D 9C570001 MOV DWORD PTR DS:[100579C],EDI
01003720 . A3 30530001 MOV DWORD PTR DS:[1005330],EAX
01003725 . A3 94510001 MOV DWORD PTR DS:[1005194],EAX
0100372A . 893D A4570001 MOV DWORD PTR DS:[10057A4],EDI
01003730 . 890D A0570001 MOV DWORD PTR DS:[10057A0],ECX
01003736 . C705 00500001 MOV DWORD PTR DS:[1005000],1
01003740 . E8 25FDFFFF CALL winmine.0100346A
01003745 . 53 PUSH EBX Arg1
01003746 . E8 05E2FFFF CALL winmine.01001950 winmine.01001950
0100374B . 5F POP EDI
0100374C . 5E POP ESI
0100374D . 5B POP EBX
0100374E . C3 RETN
0100374F $ 53 PUSH EBX

Constants

- Constants must fit into the 32-bit instruction width
- Can't therefore be the full 32-bits
- 8-bit + a 4-bit shift — gives a wide range of values
- Also a Move Negated instruction
- If that doesn't work, load from a literal pool

ARM Speed

- Does running at 1GHz means its going to be slow?
- GHz-speed tells us ‘cycles per second’
  - Actual speed depends on how many cycles an instruction takes
  - ARM aims for one-cycle per instruction
- x86 instructions can take many cycles
  - If an ARM instruction takes 1 cycle at 1GHz
  - And an x86 instruction takes 3 cycles at 3GHz
  - Which is faster?
- Speed is not entirely down to clock speed
  - It’s what you do with it

ARM Conditional Execution

- Often want to conditionally execute some code – while loops, for loops etc

```
cmp al,dl
jg label1
```

  - Traditional approach is to execute a compare
  - Branch if the condition is met
  - Branches are 'expensive' (difficult to make parallel efficiently)
- ARM allows any instruction to made conditional
  - Remove the need for an expensive branch
  - Smaller code footprint

ARM and Thumb

- Every ARM instruction is 32bits long
  - Can take up a lot of memory
  - Memory accesses take time
  - Can slow things down
- Thumb
  - 16-bit version of the ARM instruction set
  - Variable length instruction set
  - Took the most popular ARM instructions used and encoded them as 16-bit values
  - Why?

ARM and Thumb

- Speed
  - ARM instructions require 32bits to be read every instruction
  - Memory reads take time
  - Not all memory is 32bit wide
    - On smaller RAM width, takes multiple reads to get an instruction
- By making the instructions smaller gain a speed up
  - 8-bit memory two reads per instruction
  - 16-bit memory, one read per instruction
  - 32-bit memory, one read gives two instructions
- iPhone SDK generates Thumb by default

Floating Point

- Thumb doesn't encode FPU instructions
- CPU would have to branch to ARM instructions to execute it
  - This takes time
- FPU heavy code is better compiled to ARM, not Thumb
  - Assuming the device has an FPU!

ARM big.LITTLE

- ARM's latest processor contains two cores
  - 2.0GHz quad core optimised for performance (big core)
  - 1.0GHz quad core optimised for energy efficiency (LITTLE core)
- Two Cores are architecturally consistent
- System can switch between the two as appropriate for the task in hand

Next time

- Introduction to Android development tools
- Android as an operating system
  - What is it?