Građa računala

ARM i ARM arhitektura

Instruction Set arhitektura

- Izvršavanje viših jezika direktno
- Izvršavanje kompleksnih instrukcija (CISC)
- “Složiti” instruction set za implementaciju visoko-performantnog pipelinea. “Pokazati” pipeline za instrukcije kompajleru radi optimizacije koda I pojednostavljenja hardvera (RISC)
- Dati dodatne eksplicitne informacije o međuovisnostima između instrukcija (VLIW ili slično)

Instruction Set Architecture

- Najbolji IS je onaj koji daje najbolju implementaciju
- Promjene IS su teške I ne dešavaju se često
- Faktori koji utječu na dizajn IS se mijenjaju s vremenom, npr. aplikacije, programski jezici, tehnologije kompajlera, budžeti, proizvodna tehnologija
- Ne trebamo uključivati mogućnosti besmislene dodatne mogućnosti

RISC pristup

- Sve uobičajene primjene napraviti jako brzima kroz odabir najkorisnijih instrukcija, vrsta adresiranja I sl.
- Instrukcije dizajnirane da dobro koriste register file
- RISC ISA je dizajniran za jednostavnu implementaciju visokih performansi

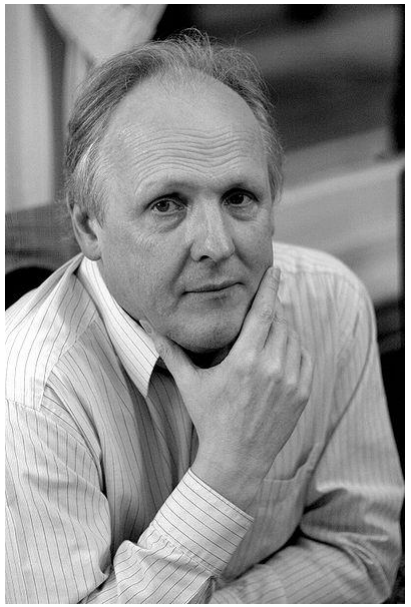
Instruction Set Architecture

Uobičajene mogućnosti:

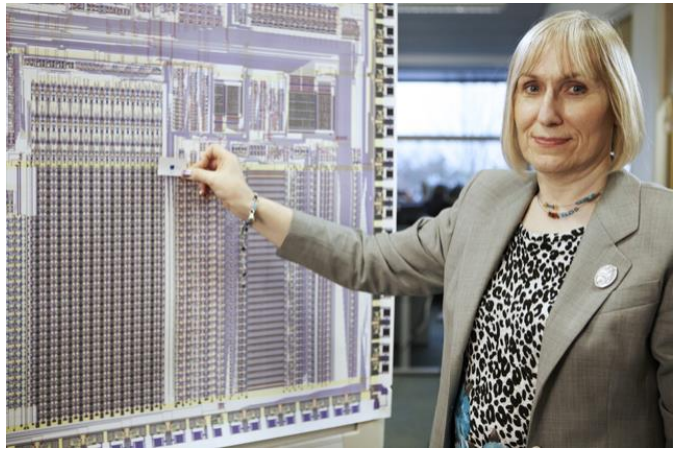
- Instrukcije fiksne duljine (ili manji broj formata instrukcija koje je jednostavno dekodirati)
- Svaka instrukcija slijedi slične korake za vrijeme izvršavanja
- Pristup podatkovnoj memoriji je ograničen na specijalne load/store instrukcije (za load-store arhitekturu)

Arm1: Prvi ARM procesor (1985)

- Arm: **A**dvanced **R**ISC **M**achine (Arm)
- Dizajnirali su ga Sophie Wilson and Prof. Steve Furber, bio je inspiriran istraživačkim radovima sa Berkley-a I Stanford-a na temu RISC arhitekture
- Arm1
  - 25,000 tranzistora
  - 3-stage pipeline
  - 8 MHz clock
  - No on-chip cache

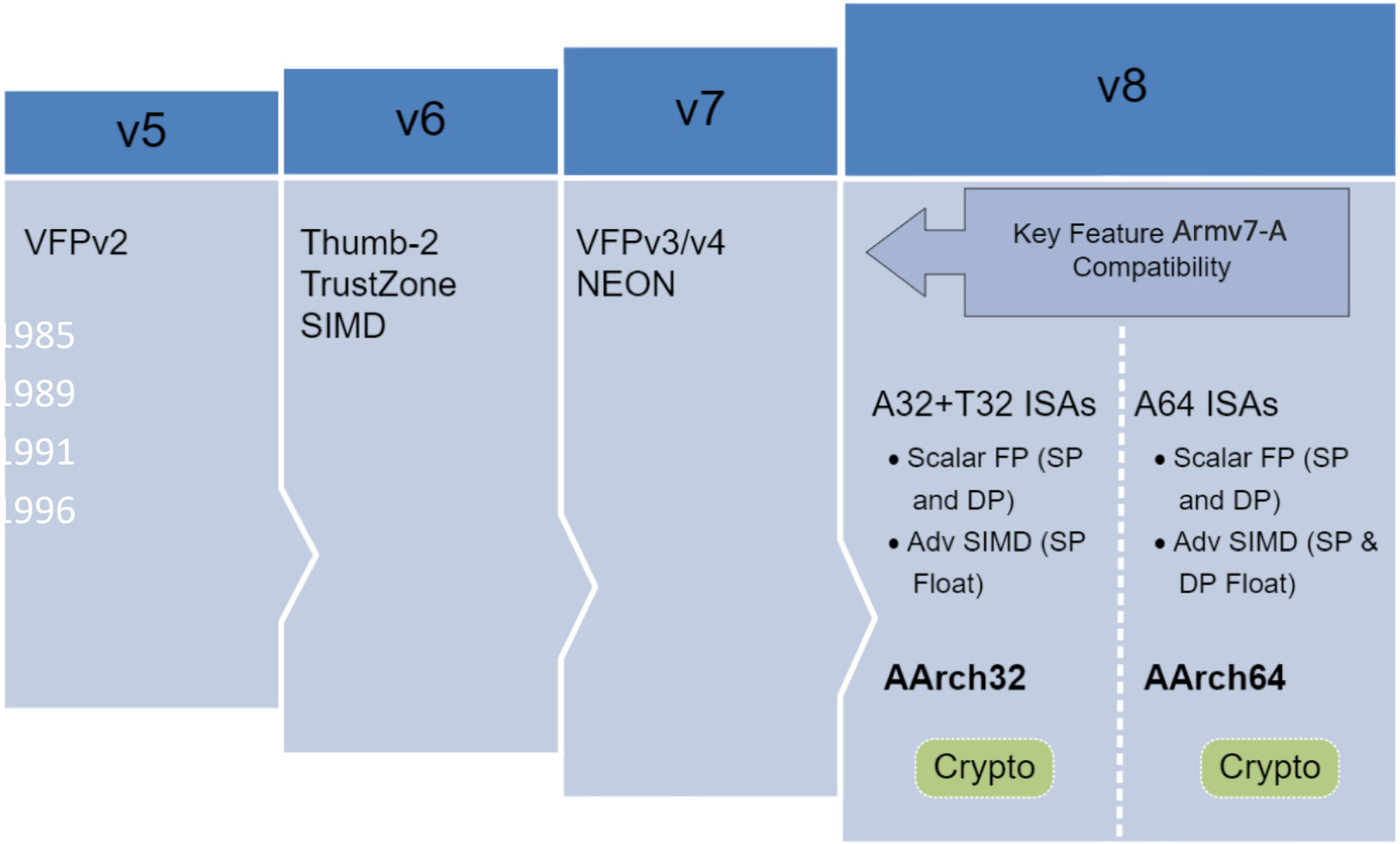


Prof. Steve Furber



Sophie Wilson

Armv8 Architecture



Announced 2011

AArch64 – Koliko se razlikuje od starijih ARM ISA-a?

- Gotovo potpuno eliminirano uvjetno izvršavanje
- Nema posmicanja
- PC nije dio integer register seta
- Nema mogućnosti za load-store više instrukcija istovremeno
- Puno jednostavnije kodiranje instrukcija

A64 instrukcije

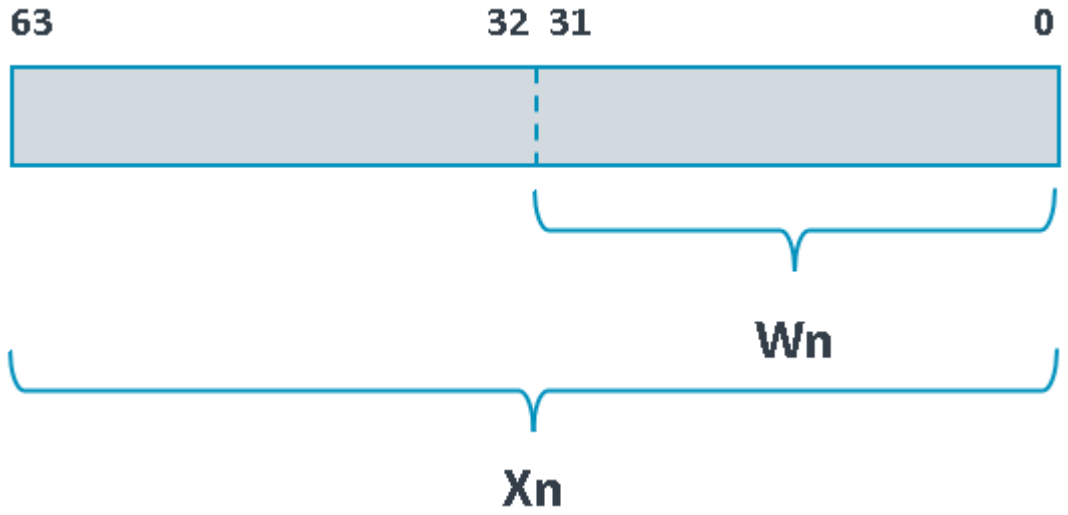
- 64-bit pointeri I registri
- Fixed-length 32-bit instrukcije
- Load/store arhitektura
- Jednostavni načini adresiranja
- 32x64-bit registara opće namjene (uključujući I R31, zero/stack registar)
- PC ne može biti specificiran kao odredište procesiranja podataka ili load instrukcije

AArch64 - registri

U Execution fazi, svaki register (X0-X30) je širine 64 bita. Povećana širina (vs 32 bita) pomaže da se smanji pritisak na register u većini aplikacija

Svaki 64-bit registar opće namjene (X0-X30) također ima I 32-bitnu formu (W0 - W30)

Zero register – X31



AArch64 – Load/Store instrukcije

LDR – load data from an address into a register.

STR – store data from a register to an address.

```
LDR X0, <addr>  ; load from <addr> into X0
STR X0, <addr>  ; store contents of X0 to <addr>
```

U ovim slučajevima, X0 je 64-bitni registar, pa će se 64 bita učitavati ili snimati iz/u memoriju.

AArch64 – načini adresiranja 1

Base register only: Address to load/store from is a 64-bit base register.

```
LDR X0, [X1]            ; load from address held in X1
STR X0, [X1]            ; store to address held in X1
```

Base plus offset: We can add an immediate or register offset (**register indexed**).

```
LDR X0, [X1, #8]        ; load from address [X1 + 8 bytes]
LDR X0, [X1, #-8]       ; load from address [X1 - 8 bytes]
LDR X0, [X1, X2]        ; load from address [X1 + X2]
LDR X0, [X1, X2, LSL #3] ; left-shift X2 three places
                            before adding to X1
```

•

AArch64 – načini adresiranja 2

Pre-indexed: source register changed before load

```
LDR W0, [X1, #4]!    ; equivalent to:
                       ADD X1, X1, #4
                       LDR W0, [X1]
```

Post-indexed: source register changed after load

```
LDR W0, [X1], #4     ; equivalent to:
                       LDR W0, [X1]
                       ADD X1, X1, #4
```

AArch64 – procesiranje podataka

- Vrijednosti u registrima procesiramo kroz različite instrukcije:
  - Aritmetičke, logičke, seljenje podataka, manipulacije bitovima, posmicanje, uvjetne usporedbe I sl.
- Ove instrukcije uvijek rade između registara

Example loop:

```
        MOV X0, #<loop count>
Loop:
        LDR W1, [X2]
        ADD W1, W1, W3
        STR W1, [X2], #4
        SUB X0, X0, #1
        CBNZ X0, loop
```

AArch64 - Branching

`B <offset>`

PC relative branch (+/- 128MB)

`BL <offset>`

Similar to B, but also stores return address in LR (link register), likely a function call

`BR Xm`

Absolute branch to address stored in **Xm**

`BRL Xm`

Similar to BR, but also stores return address in LR

AArch64 - Branching

`RET Xm` ili `RET`

- Slično kao BR, obično kao povrat u funkciju
- Koristi LR ako ne specificiramo registar

Pozivi podrutina:

LR (Link Register) posprema return adresu kada se pozove podrutina. Kasnije se to koristi na kraju rutine za povratak na iduću instrukciju.

AArch64 – uvjetno izvršavanje

A64 set instrukcija ne uključuje koncept široko rasprostranjenog prediciranog ili uvjetnog izvršenja (kao što su to činili raniji Arm ISA).

NZCV registar sadrži kopije zastavica stanja N, Z, C i V.

Osiguran je mali skup uvjetnih uputa za obradu podataka koje koriste zastavice uvjeta kao dodatni ulaz. Uvjetno se izvršava samo uvjetna grana.

- Uvjetno grananje
- Dodaj/oduzmi s prijenosom
- Uvjetni odabir s povećanjem, negiranjem ili invertiranjem
- Uvjetna usporedba (uz postavljanje zastavice uvjeta)

AArch64 – uvjetni branchevi

B.cond

Branch to label if condition code evaluates to true, e.g.,

```
CMP X0, #5
B.EQ label
```

CBZ/CBNZ – branch to label if operand register is zero (CBZ) or not equal to zero (CBNZ)

TBZ/TBNZ – branch to label if specific bit in operand register is set (TBZ) or clear (TBNZ)

```
TBZ W0, #20, label ; branch if (W0[20]==#0b0)
```

AArch64 – operatori uvjeta

`CSEL` – select between two registers based on a condition

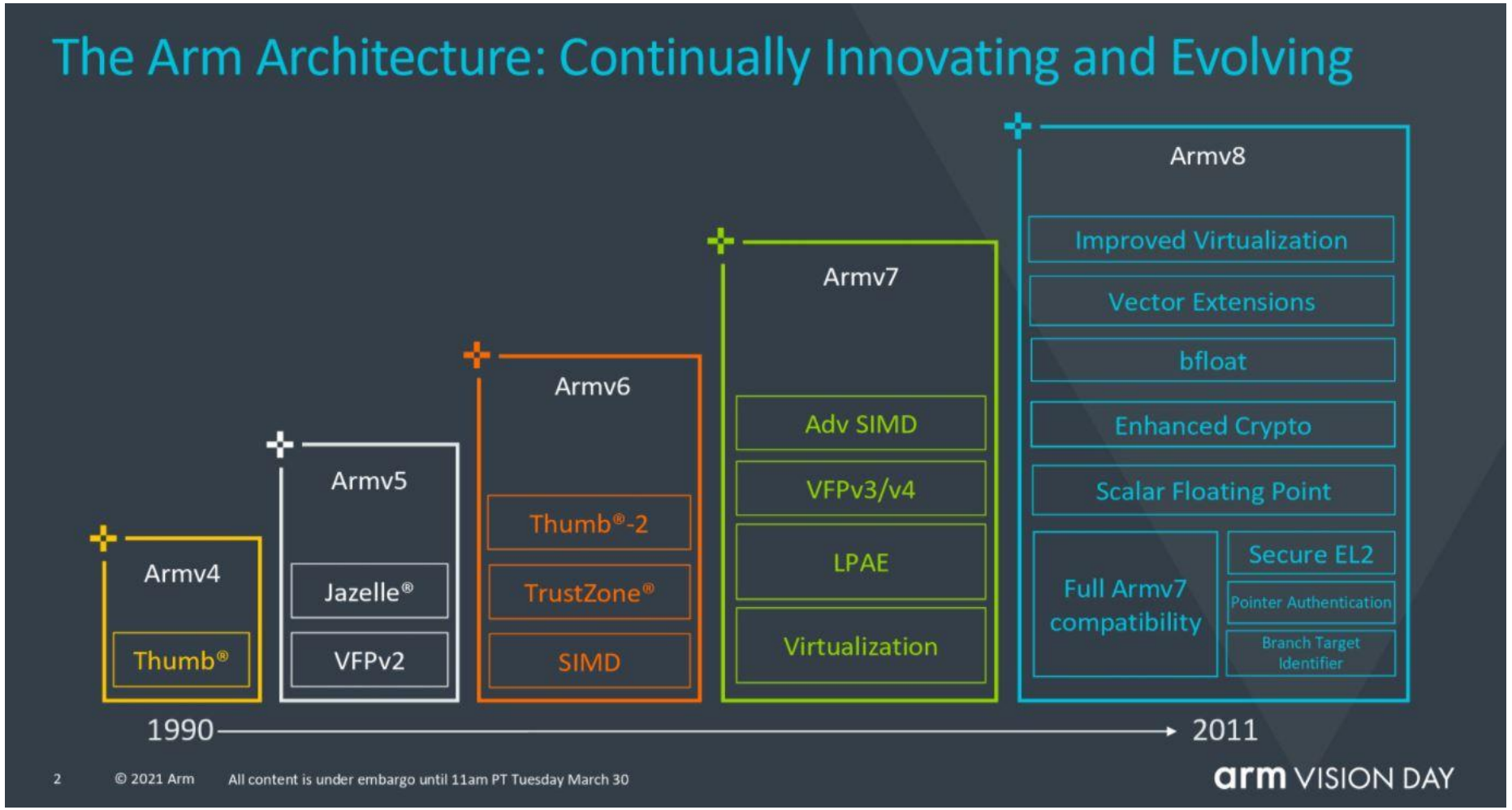
```
CSEL X7, X2, X0, EQ ; if (cond==true) X7=X2, else X7=X0
```

There are also variants of this that cause the second source register to be incremented, inverted, or negated.

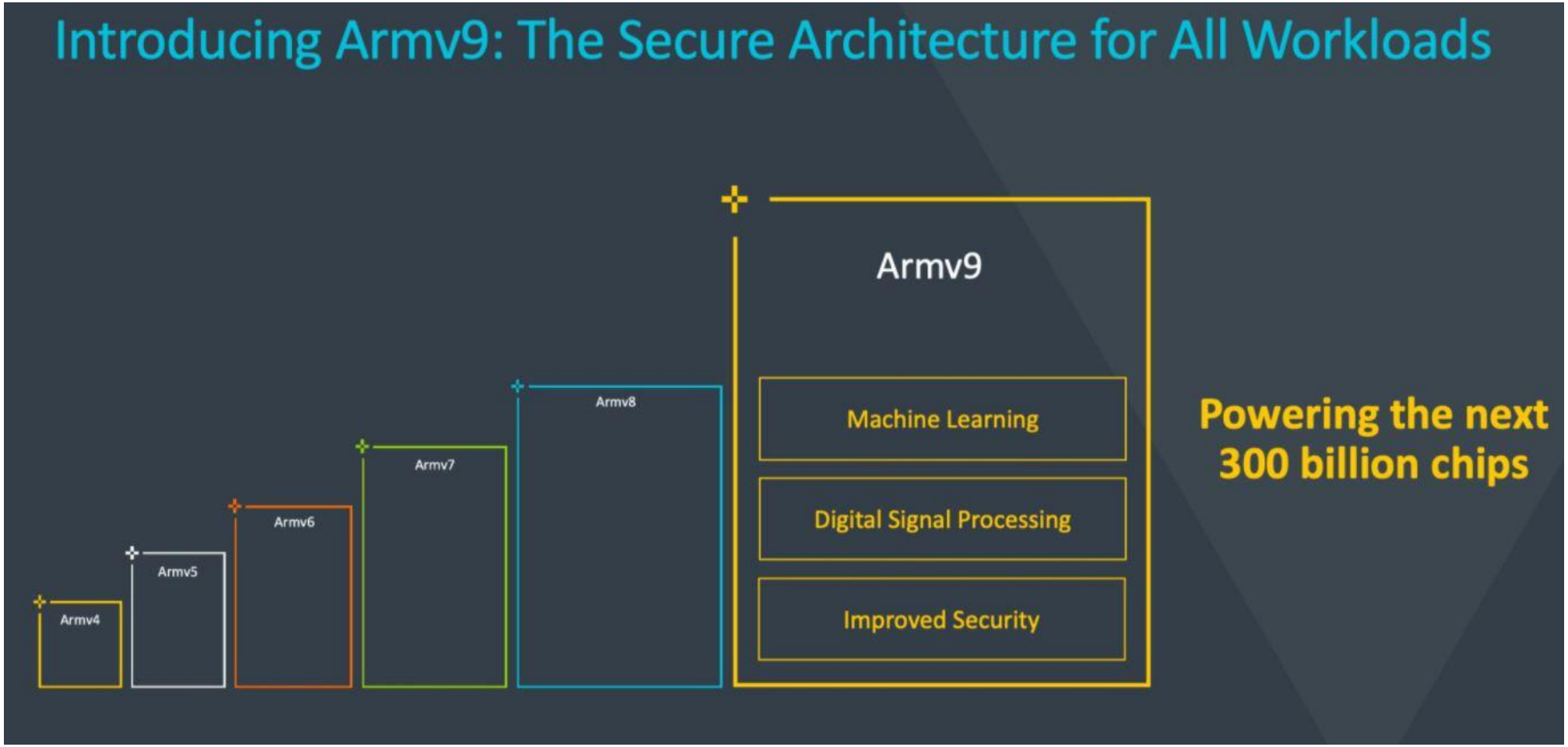
Armv9 arhitektura

- Predstavljena 2021.
- Fokus na:
  - security (Arm CCA, Confidential Compute Architecture)
  - AI, SVE2 (Scalable Vector Extension)
  - Compute dizajn
- Na neki način, moglo bi se reći da je Armv8 bio pomalo orijentiran na stolno računalo (ARM dizajni za stolna računala i poslužitelje to dokazuju)
- Na neki način, moglo bi se reći da Armv9 više naginje superračunalstvu, u arhitektonskom smislu – za SC, za HPC
- Razlog je vrlo jednostavan – zahtjevi i potrebe dizajna, o čemu smo već više puta raspravljali
- To ćemo dodatno demonstrirati dok se krećemo prema paralelizmu i pamćenju, jer kada obradimo te teme, postat će još očitije zašto

Vizualizacija ARM povijesti, part I



Vizualizacija ARM povijesti, part II



Hvala na pažnji!