U N I V E R S I D A D E DE S A˜ O P A U L O

**Escola de Artes, Ciˆencias e Humanidades**

Relato´rio Te´cnico PPgSI-000/2019

*Arquitetura ARMv8-A*

Bruno Lourenc¸o da Cunha Kevin Gabriel Gonc¸alves de Oliveira

Renan Ernesto Silva Pinto Vinicius Alves Matias

Novembro - 2019

O conteu´do do presente relato´rio e´ de u´nica responsabilidade dos autores.

Se´rie de Relato´rios Te´cnicos PPgSI-EACH-USP

Rua Arlindo Be´ttio, 1000 – Ermelino Matarazzo 03828-000 – Sa˜o Paulo, SP.

TEL: (11) 3091-8197 <http://www.each.usp.br/ppgsi>

**Arquitetura ARMv8-A**

## Bruno Lourenc¸o da Cunha1 , Kevin Gabriel Gonc¸alves de Oliveira2 , Renan Ernesto Silva Pinto3 , Vinicius Alves Matias4

1Escola de Artes, Cieˆncias e Humanidades – Universidade de Sa˜o Paulo Sa˜o Paulo – SP, Brazil

bruno [cunha@usp.br](mailto:cunha@usp.br)

2Escola de Artes, Cieˆncias e Humanidades – Universidade de Sa˜o Paulo Sa˜o Paulo – SP, Brazil

[autor2@email.com](mailto:autor2@email.com)

3Escola de Artes, Cieˆncias e Humanidades – Universidade de Sa˜o Paulo Sa˜o Paulo – SP, Brazil

[renan.ernesto@usp.br](mailto:renan.ernesto@usp.br)

4Escola de Artes, Cieˆncias e Humanidades – Universidade de Sa˜o Paulo Sa˜o Paulo – SP, Brazil

[viniciusmatias@usp.br](mailto:viniciusmatias@usp.br)

***Resumo.*** *Relato´rio te´cnico referente a` arquitetura ARMv8-A, desenvolvida pela ARM Holdings e presente em diversos chips fabricados para dispositivos mo´veis.*

# Histo´rico

A arquitetura ARMv8-A foi anunciada em outubro de 2011 pela empresa britaˆnica ARM (Advanced RISC Machine) Holdings como a oitava versa˜o da arquitetura ARM, tendo seu perfil definido para aplicac¸o˜es e portanto, sendo otimizada para sistemas ope- racionais de alto n´ıvel. A arquitetura pertencente a` linha de arquitetura RISC (Reduced Instruction Set Computer) contendo as seguintes caracter´ısticas:

Um grande arquivo uniforme de registro.

•

Uma arquitetura de *load/store*, onde as operac¸o˜es de processamento de dados operam somente no conteu´do do registrador e na˜o diretamente no conteu´do da memo´ria.

•

Modo simples de enderec¸amento, com todos os enderec¸os de *load/store* determi- nados somente do conteu´do do registrador e campos da intruc¸a˜o.

•

A arquitetura suporta tanto enderec¸amento quanto aritme´tica de 64 bits e instruc¸o˜es de tamanho fixo de 32 bits, ale´m de um estado de execuc¸a˜o de 64 bits (AArch64) e outro

de 32 bits (AArch32), que e´ completamente compat´ıvel com as verso˜es anteriores da

arquitetura ARM. Atualmente, a arquitetura vem sendo aprimorada e esta´ em constante evoluc¸a˜o. A ARMv8.6-A fornece um ambiente prop´ıcio para o desenvolvimento de Redes Neurais (NN) para Machine Learning (ML) atrave´s de General Matrix Multiply (GEMM) e BFloat 16. Ale´m de todas evoluc¸o˜es presentes nas verso˜es ARMv8.1-A, ARMv8.2-A, ARMv8.3-A, ARMv8.4-A e ARMv8.5-A.

# Uso atual

A arquitetura esta´ presente em diversos chips que visam uma boa eficieˆncia energe´tica aliada a um alto desempenho. Um exemplo de chip com tais caracter´ısticas e´ o Snap- dragon 855, presente em celulares como Asus Zenfone 5Z1, Xiaomi Mi 82 e va´rios ou- tros3 que usam as microarquiteturas Cortex-A76 e Cortex-A55. Com o encerramento da fabricac¸a˜o dos processadores Atom da Intel (maio de 2016), a arquitetura ARMv8-A tornou-se o padra˜o da indu´stria para todos os dispositivos mo´veis.

# Desempenho

Para medir o desempenho da arquitetura, o dispositivo usado como refereˆncia foi o OnePlus 7 Pro, equipado com o chip Snapdragon 855 (anunciado em 5 de Dezembro de 2018 pela Qualcomm Technologies, baseado na arquitetura ARMv8-A). Os testes fo- ram realizados a partir da versa˜o 5.0.3 do Geekbench para Android AArch64. Como refereˆncia, o Geekbench estabelece 1000 pontos como sendo o resultado da pontuac¸a˜o de um i3-8100. Os resultados foram os seguintes:

* + Single-Core Score: 763 Pontos
  + Single-Core Crypto Score: 1027 Pontos
  + Single-Core Integer Score: 734 Pontos
  + Single-Core Floating Point Score: 781 Pontos
  + Multi-Core Score: 2778 Pontos
  + Multi-Core Crypto Score: 3974 Pontos
  + Multi-Core Integer Score: 2707 Pontos
  + Multi-Core Floating Point Score: 2731 Pontos

# Instruction Set

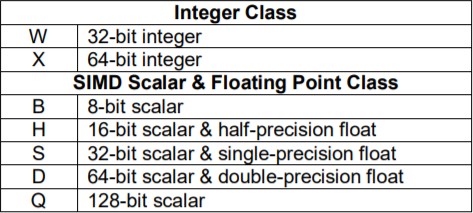
## Estrutura ba´sica

O assembler da arquitetura ARMv8-A reconhece instruc¸o˜es tanto em caixa alta como em caixa baixa. As instruc¸o˜es sa˜o linhas compostas por um ou mais ro´tulos(labels) segui- dos do nome da operac¸a˜o, um registrador de destino e um ou mais registradores, separados por v´ırgula, utilizados na operac¸a˜o. Sendo assim, a estrutura das instruc¸o˜es segue o se- guinte padra˜o:

label:\* opcode dest , source1 , source2 , source3

{ } { { { { { }}}}}

A ordem do registrador de destino e dos registradores fonte sa˜o trocadas apenas na instruc¸a˜o store.

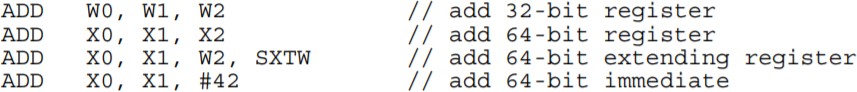
Na instruc¸a˜o assembly os registradores por sua vez podem assumir diferentes formatos. A tabela abaixo elenca isso:

1[https://www.asus.com/Phone/ZenFone-](http://www.asus.com/Phone/ZenFone-5Z-ZS620KL/Tech-Specs/)5Z-[ZS620KL/Tech-Specs/](http://www.asus.com/Phone/ZenFone-5Z-ZS620KL/Tech-Specs/) .

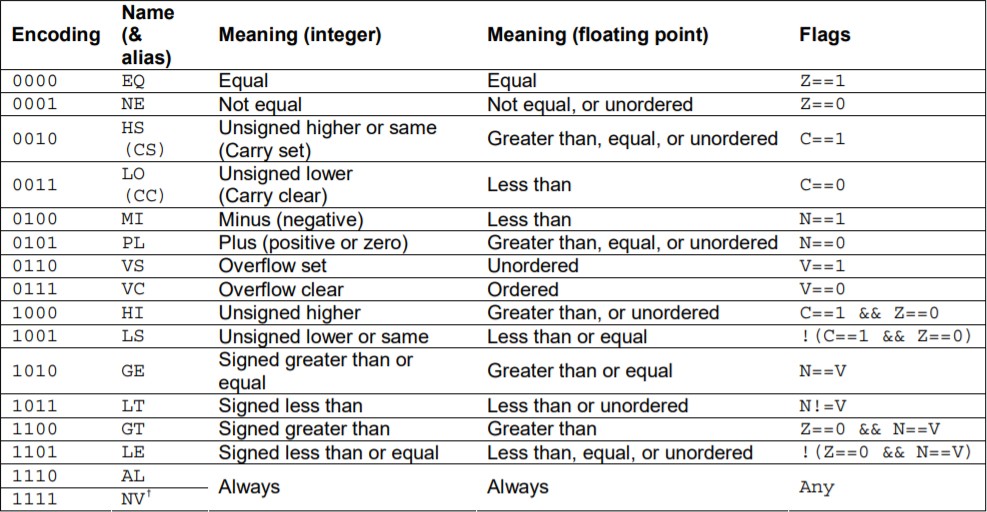
2[https://www.mi.com/global/mi8/specs](http://www.mi.com/global/mi8/specs) .

3[https://www.techwalls.com/qualcomm-snapdragon-](http://www.techwalls.com/qualcomm-snapdragon-855-smartphones/)855-[smartphones/](http://www.techwalls.com/qualcomm-snapdragon-855-smartphones/) .

Assim, as instruc¸o˜es podem tomar as seguintes formas:



## Condicionais



* 1. **Usados frequentemente**

As seguintes sintaxes sa˜o frequentemente usadas:

**Xn -** O operador Xn ou Wn interpreta o registrador 31 como um registrador zero, re- presentado pelos nomes XZR ou WZR respectivamente.

**Xn SP -** O operador Xn SP ou Wn WSP interpreta o registrador 31 como o ponteiro de pilha representado pelos nomes SP ou WSP respectivamente.

| | |

**cond -** Uma condic¸a˜o padra˜o ARM como EQ, NE, CS HS, CC LO, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE, AL ou NV com os mesmos significados da arquitetura AArch32.

| |

**invert(cond) -** O inverso de cond, por exemplo, o inverso de GT e´ LE.

**uimmn -** Um n-bit valor imediato sem sinal.

**simmn -** Um n-bit valor imediato com sinal em forma de complemento de 2 (onde n inclui o bit de sinal).

**label -** Representa uma refereˆncia para uma parte do co´digo ou localizac¸a˜o.

**addr -** Representa um modo de enderec¸amento.

**lshift -** Representa um operador de deslocamento realizado ao final de operadores lo´gicos. Engloba instruc¸o˜es como LSL, LSR, ASR ou ROR, que sa˜o seguidas de uma quantidade constante de deslocamento.

**ashift -** Representa um operador de deslocamento realizado ao final de operadores ari- time´ticos. Engloba instruc¸o˜es como LSL, LSR, ou ASR, que sa˜o seguidas de uma quan- tidade constante e deslocamento.

## Controle de Fluxo

* + 1. **Branch condicional**

Operadores utilizados em branch’s condicionais:

**B.cond label -** Branch: Condicionalmente salta para label se cond e´ verdadeiro.

**CBNZ Wn, label -** Compare and Branch Not Zero: Condicionalmente salta para label se Wn na˜o e´ igual a zero.

**CBNZ Xn, label -** Compare and Branch Not Zero (extended): Condicionalmente salta para label se Xn na˜o e´ igual a zero.

**CBZ Wn, label -** Compare and Branch Zero: Condicionalmente salta para label se Wn e´ igual a zero.

**CBZ Xn, label -** Compare and Branch Zero (extended): Condicionalmente salta para label se Xn e´ igual a zero.

**TBNZ Xn Wn #uimm6 label -** Test and Branch Not Zero: condicionalmente salta para label se o nu´mero de bits uimn6 no registrador Xn na˜o e´ igual a zero.

|

**TBZ Xn Wn, #uimm6, label -** Test and Branch Zero: condicionalmente salta para la- bel se o nu´mero de bits uimn6 no registrador Xn e´ igual a zero.

|

## Branch na˜o condicional imediato

Operadores utilizados em branch’s na˜o condicionais imediatos:

**B label -** Branch: incondicionalmente salta para label.

**BL label -** Branch and Link: incondicionalmente salta para label e escreve o enderec¸o da pro´xima instruc¸a˜o sequencial no registrador X30.

## Branch na˜o condicional com registrador

Operadores utilizados em branch’s na˜o condicionais utilizando o enderec¸o de me´moria em registradores:

**BLR Xm -** Branch and Link Register: incondicionalmente salta para o enderec¸o em Xm e escreve o enderec¸o da pro´xima instruc¸a˜o sequencial no registrador X30.

**BR Xm -** Branch Register: salta para o enderec¸o em Xm com um lembrete a CPU que isso na˜o e´ um retorno de subrotina.

**RET Xm -** Return: salta para o enderec¸o em Xm com um lembrete a CPU que isso e´ um retorno de subrotina.

## Memory Access

* + 1. **Load-Store com registrador u´nico**

A forma mais geral de load-store, ela suporta uma variedade de modos de enderec¸amento ale´m dos registradores base Xn ou SP.

**LDR Wt, addr -** Load Register: carrega em Wt uma palavra da memo´ria enderec¸ada por addr.

**LDR Xt, addr -** Load Register (extended): carrega em Xt uma dupla-palavra (dou- bleword) da memo´ria enderec¸ada por addr.

**LDRB Wt, addr -** Load Byte: carrega em Wt um byte da memo´ria enderec¸ada por addr e preenche os bits restantes com zero.

**LDRSB Wt, addr -** Load Signed Byte: carrega em Wt um byte da memo´ria enderec¸ada por addr e preenche os bits restantes de acordo com o sinal.

**LDRSB Xt, addr -** Load Signed Byte (extended): carrega em Xt um byte da memo´ria enderec¸ada por addr e preenche os bits restantes de acordo com o sinal do byte.

**LDRH Wt, addr -** Load Halfword: carrega em Wt metade de uma palavra (halfword) da memo´ria enderec¸ada por addr e preenche os bits restantes com zero.

**LDRSH Wt, addr -** Load Signed Halfword: carrega em Wt metade de uma palavra (halfword) da memo´ria enderec¸ada por addr e preenche os bits restantes de acordo com o sinal.

**LDRSH Xt, addr -** Load Signed Halfword (extended): carrega em Xt metade de uma palavra (halfword) da memo´ria enderec¸ada por addr e preenche os bits restantes de acordo com o sinal.

**LDRSW Xt, addr -** Load Signed Word (extended): carrega em Xt uma palavra da memo´ria enderec¸ada por addr e preenche os bits restantes de acordo com o sinal.

**STR Wt, addr -** Store Register: armazena uma palavra de Wt no enderec¸o de memo´ria addr.

**STR Xt, addr -** Store Register (extended): armazena uma palavra dupla (doubleword) de Xt no enderec¸o de memo´ria addr.

**STRB Wt, addr -** Store Byte: armazena um byte contido em Wt no enderec¸o de memo´ria addr.

**STRH Wt, addr -** Store Halfword: armazena metade de uma palavra (halfword) con- tida em Wt no enderec¸o de memo´ria addr.

## Data Processing (immediate)

Os seguintes grupos de instruc¸a˜o sa˜o suportados:

* Aritme´ticas (immediate)
* Lo´gicas (immediate)
* Move (immediate)
* Bitfield (operations)
* Deslocamento (immediate)
* Extensa˜o de sinal/zero

## Arithmetic (immediate)

Operac¸o˜es aritme´ticas que aceitam valores imediatos:

**ADD Wd**|**WSP, Wn**|**WSP, #aimm -** Add (immediate): Wd|WSP = Wn|WSP + aimm.

**ADD Xd**|**SP, Xn**|**SP, #aimm -** Add (extended immediate): Xd|SP = Xn|SP + aimm.

**ADDS Wd, Wn WSP, #aimm -** Add and set flags (immediate): Wd = Wn WSP + aimm, configurando as flags de condic¸a˜o.

| |

**ADDS Xd, Xn SP, #aimm -** Add and set flags (extended immediate): Xd = Xn SP + aimm, configurando as flags de condic¸a˜o.

| |

**SUB Wd**|**WSP, Wn**|**WSP, #aimm -** Subtract (immediate): Wd|WSP = Wn|WSP - aimm.

**SUB Xd**|**SP, Xn**|**SP, #aimm -** Subtract (extended immediate): Xd|SP = Xn|SP - aimm.

**SUBS Wd, Wn WSP, #aimm -** Subtract and set flags (immediate): Wd = Wn WSP - aimm, configurando as flags de condic¸a˜o.

| |

**SUBS Xd, Xn SP, #aimm -** Subtract and set flags (extended immediate): Xd = Xn SP

| |

- aimm, configurando as flags de condic¸a˜o.

**CMP Wn**|**WSP, #aimm -** Compare (immediate): pseudoˆmino para SUBS WZR,Wn|WSP,#aimm.

**CMP Xn, #aimm -** Compare (extended immediate): pseudoˆmino para SUBS XZR,Xn|SP,#aimm.

**CMN Wn**|**WSP, #aimm -** Compare negative (immediate): pseudoˆmino para ADDS WZR,Wn|WSP,#aimm.

**CMN Xn**|**SP, #aimm -** Compare negative (extended immediate): pseudoˆmino para ADDS XZR,Xn|SP,#aimm.

**MOV Wd WSP, Wn WSP -** Move (register): pseudoˆmino para ADD Wd WSP,Wn WSP,#0, mas somente quando algum dos registradores e´ o WSP. Em outros casos a instruc¸a˜o ORR Wd,WZR,Wn e´ usada.

| | | |

**MOV Xd SP, Xn SP -** Move (extended register): pseudoˆmino para ADD Xd SP,Xn SP,#0, mas somente quando algum dos registradores e´ o SP. Em outros casos a instruc¸a˜o ORR Xd,XZR,Xn e´ usada.

| | | |

## Logical (immediate)

Operac¸o˜es lo´gicas que aceitam valores imediatos:

**AND Wd WSP, Wn, #bimm32 -** Bitwise AND (immediate): Wd WSP = Wn AND bimm32.

| |

**AND Xd SP, Xn, #bimm64 -** Bitwise AND (extended immediate): Xd SP = Xn AND bimm64.

| |

**ANDS Wd, Wn, #bimm32 -** Bitwise AND and Set Flags (immediate): Wd = Wn AND bimm32, atribuindo as flags de condic¸a˜o N e Z baseado no resultado, ale´m de limpar as flags C e V.

**ANDS Xd, Xn, #bimm64 -** Bitwise AND and Set Flags (extended immediate): Xd = Xn AND bimm64, atribuindo as flags de condic¸a˜o N e Z baseado no resultado, ale´m de limpar as flags C e V.

**EOR Wd WSP, Wn, #bimm32 -** Bitwise exclusive OR (immediate): Wd WSP = Wn EOR bimm32.

| |

**EOR Xd SP, Xn, #bimm64 -** Bitwise exclusive OR (extended immediate): Xd SP = Xn EOR bimm64.

| |

**ORR Wd WSP, Wn, #bimm32 -** Bitwise inclusive OR (immediate): Wd WSP = Wn OR bimm32.

| |

**ORR Xd SP, Xn, #bimm64 -** Bitwise inclusive OR (extended immediate): Xd SP = Xn OR bimm64.

| |

**MOVI Wd, #bimm32 -** Move bitmask (immediate): pseudoˆmino para ORR Wd,WZR,#bimm32.

**MOVI Xd, #bimm64 -** Move bitmask (extended immediate): pseudoˆmino para ORR

Xd,XZR,#bimm64.

**TST Wn, #bimm32 -** Bitwise test (immediate): pseudoˆmino para ANDS WZR,Wn,#bimm32.

**TST Xn, #bimm64 -** Bitwise test (extended immediate): pseudoˆmino para ANDS XZR,Xn,#bimm64

## Move (wide immediate)

As seguintes instruc¸o˜es inserem um valor imediato em um registrador destino

**MOVZ Wt, #uimm16, LSL #pos -** Move with Zero (immediate): Wt = LSL(uimm16, pos).

**MOVZ Xt, #uimm16, LSL #pos -** Move with Zero (extended immediate): Xt = LSL(uimm16, pos).

**MOVN Wt, #uimm16, LSL #pos -** Move with NOT (immediate): Wt = NOT(LSL(uimm16, pos)).

**MOVN Xt, #uimm16, LSL #pos -** Move with NOT (extended immediate): Xt = NOT(LSL(uimm16, pos)).

**MOVK Wt, #uimm16, LSL #pos -** Move with Keep (immediate): Wt¡pos+15:pos¿ = uimm16.

**MOVK Xt, #uimm16, LSL #pos -** Move with Keep (extended immediate): Xt¡pos+15:pos¿

= uimm16.

**MOV Wd, #simm32 -** Uma instruc¸a˜o sinte´tica que gera tanto MOVZ, MOVN, como MOVI.

**MOV Xd, #simm64 -** Funciona assim como o MOV pore´m para carregar registradores Xd com valores de 64-bit.

## Bitfield Operations

Operac¸o˜es de bits:

**BFM Wd, Wn, #r, #s -** Bitfield Move: se s¿=r enta˜o Wd¡s-r:0¿ = Wn¡s:r¿, caso contra´rio Wd¡32+s-r,32-r¿ = Wn¡s:0¿. Deixando os outros bits em Wd inalterados.

**BFM Xd, Xn, #r, #s -** Bitfield Move: se s¿=r enta˜o Xd¡s-r:0¿ = Xn¡s:r¿, caso contra´rio Xd¡64+s-r,64-r¿ = Xn¡s:0¿. Deixando os outros bits em Xd inalterados.

**SBFM Wd, Wn, #r, #s -** Signed Bitfield Move: se s¿=r enta˜o Wd¡s-r:0¿ = Wn¡s:r¿, caso contra´rio Wd¡32+s-r,32-r¿ = Wn¡s:0¿. atribuindo os bits a` esquerda com o valor do bit mais a esquerda e os bits da direita do bitfield de destino com zero.

**SBFM Xd, Xn, #r, #s -** Signed Bitfield Move: se s¿=r enta˜o Xd¡s-r:0¿ = Xn¡s:r¿, caso contra´rio Xd¡64+s-r,64-r¿ = Xn¡s:0¿. atribuindo os bits a` esquerda com o valor do bit mais a esquerda e os bits da direita do bitfield de destino com zero.

**UBFM Wd, Wn, #r, #s -** Unsigned Bitfield Move: se s¿=r enta˜o Wd¡s-r:0¿ = Wn¡s:r¿, caso contra´rio Wd¡32+s-r,32-r¿ = Wn¡s:0¿. Atribuindo zero aos bits a esquerda e a direita do bitfield de destino.

**UBFM Xd, Xn, #r, #s -** Unsigned Bitfield Move: se s¿=r enta˜o Xd¡s-r:0¿ = Xn¡s:r¿, caso contra´rio Xd¡32+s-r,32-r¿ = Xn¡s:0¿. Atribuindo zero aos bits a esquerda e a direita do bitfield de destino.

As seguintes instruc¸o˜es sa˜o pseudoˆnimos mais amiga´veis para operac¸o˜es de bit:

**BFI Wd, Wn, #lsb, #width -** Bitfield Insert: pseudoˆnimo para BFM Wd,Wn,#((32- lsb)&31),#(width-1).

**BFI Xd, Xn, #lsb, #width -** Bitfield Insert (extended): pseudoˆnimo para BFM Xd,Xn,#((64- lsb)&63),#(width-1).

**BFXIL Wd, Wn, #lsb, #width -** Bitfield Extract and Insert Low: pseudoˆnimo para BFM Wd,Wn,#lsb,#(lsb+width-1).

**BFXIL Xd, Xn, #lsb, #width -** Bitfield Extract and Insert Low (extended): pseudoˆnimo para BFM Xd,Xn,#lsb,#(lsb+width-1).

**SBFIZ Wd, Wn, #lsb, #width -** Signed Bitfield Insert in Zero: pseudoˆnimo para SBFM Wd,Wn,#((32-lsb)&31),#(width-1).

**SBFIZ Xd, Xn, #lsb, #width -** Signed Bitfield Insert in Zero (extended): pseudoˆnimo para SBFM Xd,Xn,#((64-lsb)&63),#(width-1).

**SBFX Wd, Wn, #lsb, #width -** Signed Bitfield Extract: pseudoˆnimo para SBFM Wd,Wn,#lsb,#(lsb+width- 1).

**SBFX Xd, Xn, #lsb, #width -** Signed Bitfield Extract (extended): pseudoˆnimo para SBFM Xd,Xn,#lsb,#(lsb+width-1).

**UBFIZ Wd, Wn, #lsb, #width -** Unsigned Bitfield Insert in Zero: pseudoˆnimo para UBFM Wd,Wn,#((32-lsb)&31),#(width-1).

**UBFIZ Xd, Xn, #lsb, #width -** Unsigned Bitfield Insert in Zero (extended): pseudoˆnimo para UBFM Xd,Xn,#((64-lsb)&63),#(width-1).

**UBFX Wd, Wn, #lsb, #width -** Unsigned Bitfield Extract: pseudoˆnimo para UBFM

Wd,Wn,#lsb,#(lsb+width-1).

**UBFX Xd, Xn, #lsb, #width -** Unsigned Bitfield Extract (extended): pseudoˆnimo para UBFM Xd,Xn,#lsb,#(lsb+width-1).

## Extract (immediate)

Operac¸o˜es de extrac¸a˜o de bit de valores imediatos:

**EXTR Wd, Wn, Wm, #lsb -** Extract: Wd = Wn:Wm¡lsb+31,lsb¿. O bit na posic¸a˜o lsb deve estar entre 0 e 31.

**EXTR Xd, Xn, Xm, #lsb -** Extract (extended): Xd = Xn:Xm¡lsb+63,lsb¿. O bit na posic¸a˜o lsb deve estar entre 0 e 63.

## Shift (immediate)

a

**ASR Wd, Wn, uimm -** Arithmetic Shift Right (immediate): alias for SBFM Wd,Wn,uimm,31.

**ASR Xd, Xn, uimm -** Arithmetic Shift Right (extended immediate): alias for SBFM Xd,Xn,uimm,63.

**LSL Wd, Wn, uimm -** Logical Shift Left (immediate): alias for UBFM Wd,Wn,((32- uimm)&31),(31-uimm).

**LSL Xd, Xn, uimm -** Logical Shift Left (extended immediate): alias for UBFM Xd,Xn,((64- uimm)&63),(63-uimm)

**LSR Wd, Wn, uimm -** Logical Shift Left (immediate): alias for UBFM Wd,Wn,uimm,31.

**LSR Xd, Xn, uimm -** Logical Shift Left (extended immediate): alias for UBFM Xd,Xn,uimm,31.

**ROR Wd, Wm, uimm -** Rotate Right (immediate): alias for EXTR Wd,Wm,Wm,uimm.

**ROR Xd, Xm, uimm -** Rotate Right (extended immediate): alias for EXTR Xd,Xm,Xm,uimm.

## Sign/Zero Extend

**SXT[BH] Wd, Wn -** Signed Extend Byte—Halfword: alias for SBFM Wd,Wn,0,7—15.

**SXT[BHW] Xd, Wn -** Signed Extend Byte—Halfword—Word (extended): alias for SBFM Xd,Xn,0,7—15—31.

**UXT[BH] Wd, Wn -** Unsigned Extend Byte—Halfword: alias for UBFM Wd,Wn,0,7—15.

**UXT[BHW] Xd, Wn -** Unsigned Extend Byte—Halfword—Word (extended): alias for UBFM Xd,Xn,0,7—15—31.

## Data Processing (register)

a

## Arithmetic (shifted register)

**ADD Wd, Wn, Wm, ashift imm -** Add (register): Wd = Wn + ashift(Wm, imm).

**ADD Xd, Xn, Xm, ashift imm -** Add (extended register): Xd = Xn + ashift(Xm, imm).

**ADDS Wd, Wn, Wm, ashift imm -** Add and Set Flags (register): Wd = Wn + ashift(Wm, imm), setting condition flags.

**ADDS Xd, Xn, Xm, ashift imm -** Add and Set Flags (extended register): Xd = Xn + ashift(Xm, imm), setting condition flags.

**SUB Wd, Wn, Wm, ashift imm -** Subtract (register): Wd = Wn - ashift(Wm, imm).

**SUB Xd, Xn, Xm, ashift imm -** Subtract (extended register): Xd = Xn - ashift(Xm, imm).

**SUBS Wd, Wn, Wm, ashift imm -** Subtract and Set Flags (register): Wd = Wn - ashift(Wm, imm), setting condition flags.

**SUBS Xd, Xn, Xm, ashift imm -** Subtract and Set Flags (extended register): Xd = Xn - ashift(Xm, imm), setting condition flags.

**CMN Wn, Wm, ashift imm -** Compare Negative (register): alias for ADDS WZR, Wn, Wm, ashift imm.

**CMN Xn, Xm, ashift imm -** Compare Negative (extended register): alias for ADDS XZR, Xn, Xm, ashift imm.

**CMP Wn, Wm, ashift imm -** Compare (register): alias for SUBS WZR, Wn, Wm,ashift imm.

**CMP Xn, Xm, ashift imm -** Compare (extended register): alias for SUBS XZR, Xn, Xm, ashift imm.

**NEG Wd, Wm, ashift imm -** Negate: alias for SUB Wd, WZR, Wm, ashift imm.

**NEG Xd, Xm, ashift imm -** Negate (extended): alias for SUB Xd, XZR, Xm, ashift imm.

**NEGS Wd, Wm, ashift imm -** Negate and Set Flags: alias for SUBS Wd, WZR,

Wm, ashift imm.

**NEGS Xd, Xm, ashift imm -** Negate and Set Flags (extended): alias for SUBS Xd, XZR, Xm, ashift imm.

## Arithmetic (extending register)

a

**ADD Wd—WSP, Wn—WSP, Wm, extend imm -** Add (register, extending): Wd—WSP

= Wn—WSP + LSL(extend(Wm),imm).

**ADD Xd—SP, Xn—SP, Wm, extend imm -** Add (extended register, extending): Xd—SP = Xn—SP + LSL(extend(Wm),imm).

**ADD Xd—SP, Xn—SP, Xm, UXTX—LSL imm -** Add (extended register, exten- ding): Xd—SP = Xn—SP + LSL(Xm,imm).

**ADDS Wd, Wn—WSP, Wm, extend imm -** Add and Set Flags (register, extending): Wd = Wn—WSP + LSL(extend(Wm),imm), setting the condition flags.

**ADDS Xd, Xn—SP, Wm, extend imm -** Add and Set Flags (extended register, exten- ding): Xd = Xn—SP + LSL(extend(Wm),imm), setting the condition flags.

**ADDS Xd, Xn—SP, Xm, UXTX—LSL imm -** Add and Set Flags (extended register, extending): Xd = Xn—SP + LSL(Xm,imm), setting the condition flags.

**SUB Wd—WSP, Wn—WSP, Wm, extend imm -** Subtract (register, extending): Wd—WSP

= Wn—WSP - LSL(extend(Wm),imm).

**SUB Xd—SP, Xn—SP, Wm, extend imm -** Subtract (extended register, extending): Xd—SP = Xn—SP - LSL(extend(Wm),imm).

**SUB Xd—SP, Xn—SP, Xm, UXTX—LSL imm -** Subtract (extended register, exten- ding): Xd—SP = Xn—SP - LSL(Xm,imm).

**SUBS Wd, Wn—WSP, Wm, extend imm -** Subtract and Set Flags (register, exten- ding): Wd = Wn—WSP - LSL(extend(Wm),imm), setting the condition flags.

**SUBS Xd, Xn—SP, Wm, extend imm -** Subtract and Set Flags (extended register, ex- tending): Xd = Xn—SP - LSL(extend(Wm),imm), setting the condition flags.

**SUBS Xd, Xn—SP, Xm, UXTX—LSL imm -** Subtract and Set Flags (extended register, extending): Xd = Xn—SP - LSL(Xm,imm), setting the condition flags.

**CMN Wn—WSP, Wm, extend imm -** Compare Negative (register, extending): alias for ADDS WZR,Wn,Wm,extend imm.

**CMN Xn—SP, Wm, extend imm -** Compare Negative (extended register, extending): alias for ADDS XZR,Xn,Wm,extend imm.

**CMN Xn—SP, Xm, UXTX—LSL imm -** Compare Negative (extended register, ex-

tending): alias for ADDS XZR,Xn,Xm,UXTX—LSL imm.

**CMP Wn—WSP, Wm, extend imm -** Compare (register, extending): alias for SUBS WZR,Wn,Wm,extend imm.

**CMP Xn—SP, Wm, extend imm -** Compare (extended register, extending): alias for SUBS XZR,Xn,Wm,extend imm.

**CMP Xn—SP, Xm, UXTX—LSL imm -** Compare (extended register, extending): alias for SUBS XZR,Xn,Xm,UXTX—LSL imm.

## Logical (shifted register)

a

**AND Wd, Wn, Wm, lshift imm -** Bitwise AND (register): Wd = Wn AND lshift(Wm, imm).

**AND Xd, Xn, Xm, lshift imm -** Bitwise AND (extended register): Xd = Xn AND lshift(Xm, imm).

**ANDS Wd, Wn, Wm, lshift imm -** Bitwise AND and Set Flags (register): Wd = Wn AND lshift(Wm, imm), setting N & Z condition flags based on the result and clearing the C & V flags.

**ANDS Xd, Xn, Xm, lshift imm -** Bitwise AND and Set Flags (extended register): Xd = Xn AND lshift(Xm, imm), setting N Z condition flags based on the result and clearing the C V flags.

**BIC Wd, Wn, Wm, lshift imm -** Bit Clear (register): Wd = Wn AND NOT(lshift(Wm, imm)).

**BIC Xd, Xn, Xm, lshift imm -** Bit Clear (extended register): Xd = Xn AND NOT(lshift(Xm, imm)).

**BICS Wd, Wn, Wm, lshift imm -** Bit Clear and Set Flags (register): Wd = Wn AND NOT(lshift(Wm, imm)), setting N Z condition flags based on the result and clearing the C V flags.

**BICS Xd, Xn, Xm, lshift imm -** Bit Clear and Set Flags (extended register): Xd = Xn AND NOT(lshift(Xm, imm)), setting N Z condition flags based on the result and clearing the C V flags.

**EON Wd, Wn, Wm, lshift imm -** Bitwise exclusive OR NOT (register): Wd = Wn EOR NOT(lshift(Wm, imm)).

**EON Xd, Xn, Xm, lshift imm -** Bitwise exclusive OR NOT (extended register): Xd

= Xn EOR NOT(lshift(Xm, imm)).

**EOR Wd, Wn, Wm, lshift imm -** Bitwise exclusive OR (register): Wd = Wn EOR lshift(Wm, imm).

**EOR Xd, Xn, Xm, lshift imm -** Bitwise exclusive OR (extended register): Xd = Xn EOR lshift(Xm, imm).

**ORR Wd, Wn, Wm, lshift imm -** Bitwise inclusive OR (register): Wd = Wn OR lshift(Wm, imm).

**ORR Xd, Xn, Xm, lshift imm -** Bitwise inclusive OR (extended register): Xd = Xn OR lshift(Xm, imm).

**ORN Wd, Wn, Wm, lshift imm -** Bitwise inclusive OR NOT (register): Wd = Wn OR NOT(lshift(Wm, imm)).

**ORN Xd, Xn, Xm, lshift imm -** Bitwise inclusive OR NOT (extended register): Xd

= Xn OR NOT(lshift(Xm, imm)).

**MOV Wd, Wm -** Move (register): alias for ORR Wd,WZR,Wm.

**MOV Xd, Xm -** Move (extended register): alias for ORR Xd,XZR,Xm.

**MVN Wd, Wm, lshift imm -** Move NOT (register): alias for ORN Wd,WZR,Wm,lshift imm.

**MVN Xd, Xm, lshift imm -** Move NOT (extended register): alias for ORN Xd,XZR,Xm,lshift imm.

**TST Wn, Wm, lshift imm -** Bitwise Test (register): alias for ANDS WZR,Wn,Wm,lshift imm.

**TST Xn, Xm, lshift imm -** Bitwise Test (extended register): alias for ANDS XZR,Xn,Xm,lshift imm.

## Variable Shift

a

## -

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

**-**

1. **Detalhes da Arquitetura**
   1. **Tipo de Arquitetura**

A arquitetura ARMv8-A extende a arquitetura ARMv7, tendo agora dois estados de execuc¸a˜o: 32 bits e 64 bits, mantendo a compatibilidade com a arquitetura predeces- sora e, portanto, seus fundamentos. A arquitetura ARMv8-A segue uma pipeline, com a quantidade de esta´gios variando de acordo com o processador em que a arquitetura foi implementada. A ordem de execuc¸a˜o tambe´m varia de acordo com o processador, como exemplo:

* + - Cortex-A53: Pipeline executa em ordem.
    - Cortex-A57: Superescalar, na˜o segue uma ordem espec´ıfica (Speculative Issue).

## Arquitetura do Processador

Exemplificaremos a implementac¸a˜o da arquitetura ARMv8-A em um processador Cortex-A57, lanc¸ado em janeiro de 2015, com uma quantidade de cores que varia de 1 a` 4.

* + - Esta´gios por Pipeline: 15+.
    - Velocidade do Clock: 1.5 a` 2.5 GHz em 20nm.
    - Tamanho do Cache L1 (Instruc¸a˜o): 48 KB (associativo de 3 vias).
    - Tamanho do Cache L1 (Dados): 32 KB (associativo bidirecional).
    - Tamanho do Cache L2: 512 KB a` 2 MB (associativo de 16 vias).
    - Taxa de Tranfereˆncia ma´xima de Inteiro: 4.1 a` 4.76MIPS/MHz

# Refereˆncias

ARM (2011). Arm discloses technical details of the next version of the arm archi- tecture. <http://www.arm.com/about/newsroom/arm-discloses-technical-details-of-the-> next-version-of-the-arm-architecture.php. Acesso: 09 Junho 2019.

ARM (2013). *Arm*ⓍR *Architecture Reference Manual Armv8, for Armv8-A architecture*

*profile Documentation*.

Geekbench (2019). Oneplus 7 pro. https://browser.geekbench.com/v5/cpu/592464.

Acesso: 14 Novembro 2019.

Qualcomm (2018). Qualcomm announces new flagship snapdra- gon 855 mobile platform - a new decade of 5g, ai, and xr. [https://www.qualcomm.com/news/releases/2018/12/05/qualcomm-announces-ne](http://www.qualcomm.com/news/releases/2018/12/05/qualcomm-announces-new-)w- flagship-snapdragon-855-mobile-platform-new-decade. Acesso: 14 Novembro 2019.

Shah, A. (2016). Intel’s atom architecture lives on despite mobile chip cancella- tions. [https://www](http://www.pcworld.com/article/3064716/intels-atom-architecture-to-live-on-).pcw[orld.com/article/3064716/intels-atom-architecture-to-li](http://www.pcworld.com/article/3064716/intels-atom-architecture-to-live-on-)v[e-on-](http://www.pcworld.com/article/3064716/intels-atom-architecture-to-live-on-) despite-smartphone-chip-cancellations.html. Acesso: 16 Novembro 2019.

Stephens, N. (2019). Developments in the arm a-profile architecture: Armv8.6-

a. https://community.arm.com/developer/ip-products/processors/b/processors-ip- blog/posts/arm-architecture-developments-armv8-6-a. Acesso: 09 Novembro 2019.