Unconditional Branch (B)

opcode	address
6 bits	26 bits

Conditional Branch (CB)

- Based on a condition
  - CBZ (if register value is zero, then branch)
  - CBNZ (if register value is nonzero, then branch)

opcode	address	Rt
8 bits	19 bits	5 bits

- Example: CBZ X19, Exit // go to Exit if X19 == 0

180	Exit	19
8 bits	19 bits	5 bits

```
CBZ X19, 3
```
meaning [X19] == 0, then PC will move to PC+ (3x4)

Conditional branch	compare and branch on equal 0	`CBZ X1, 25`	`if (X1 == 0) go to PC + 100`	Equal 0 test; PC-relative branch
	compare and branch on not equal 0	`CBNZ X1, 25`	`if (X1 != 0) go to PC + 100`	Not equal 0 test; PC-relative branch
	branch conditionally	`B.cond 25`	`if (condition true) go to PC + 100`	Test condition codes; if true, branch
Unconditional branch	branch	`B 2500`	`go to PC + 10000`	Branch to target address; PC-relative
	branch to register	`BR X30`	`go to X30`	For switch, procedure return
	branch with link	`BL 2500`	`X30 = PC + 4; PC + 10000`	For procedure call PC-relative

Data Transfer (D)

opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits

Data transfer	load register	`LDUR X1, [X2,40]`	`X1 = Memory[X2 + 40]`	Doubleword from memory to register
	store register	`STUR X1, [X2,40]`	`Memory[X2 + 40] = X1`	Doubleword from register to memory
	load signed word	`LDURSW X1,[X2,40]`	`X1 = Memory[X2 + 40]`	Word from memory to register
	store word	`STURW X1, [X2,40]`	`Memory[X2 + 40] = X1`	Word from register to memory
	load half	`LDURH X1, [X2,40]`	`X1 = Memory[X2 + 40]`	Halfword memory to register
	store half	`STURH X1, [X2,40]`	`Memory[X2 + 40] = X1`	Halfword register to memory
	load byte	`LDURB X1, [X2,40]`	`X1 = Memory[X2 + 40]`	Byte from memory to register
	store byte	`STURB X1, [X2,40]`	`Memory[X2 + 40] = X1`	Byte from register to memory

Register (R)

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

Logical	and	`AND X1, X2, X3`	`X1 = X2 & X3`	Three reg. operands; bit-by-bit AND
	inclusive or	`ORR X1, X2, X3`	`X1 = X2 | X3`	Three reg. operands; bit-by-bit OR
	exclusive or	`EOR X1, X2, X3`	`X1 = X2 ^ X3`	Three reg. operands; bit-by-bit XOR
	and immediate	`ANDI X1, X2, 20`	`X1 = X2 & 20`	Bit-by-bit AND reg. with constant
	inclusive or immediate	`ORRI X1, X2, 20`	`X1 = X2 | 20`	Bit-by-bit OR reg. with constant
	exclusive or immediate	`EORI X1, X2, 20`	`X1 = X2 ^ 20`	Bit-by-bit XOR reg. with constant
	logical shift left	`LSL X1, X2, 10`	`X1 = X2 << 10`	Shift left by constant
	logical shift right	`LSR X1, X2, 10`	`X1 = X2 >> 10`	Shift right by constant

Logical shift left : Shift left and fill with 0 bits
LSL by i bits multiplies by 2^i
Logical shift right: Shift right and fill with 0 bits
LSR by i bits divides by 2^i (unsigned only)

More conditions

- Condition codes, set from arithmetic instruction with S-suffix (ADDS, ADDIS, ANDS, ANDIS, SUBS, SUBIS)
  - negative (N): result had 1 in MSB
  - zero (Z): result was 0
  - overlow (V): result overflowed
  - carry (C): result had carryout from MSB
- Use subtract to set flags, then conditionally branch:
  - **B.EQ**
  - **B.NE**
  - **B.LT** (less than, signed), **B.LO** (less than, unsigned)
  - **B.LE** (less than or equal, signed), **B.LS** (less than or equal, unsigned)
  - **B.GT** (greater than, signed), **B.HI** (greater than, unsigned)
  - **B.GE** (greater than or equal, signed),
  - **B.HS** (greater than or equal, unsigned)