### N-TEA (Non-Trivial Educational Architecture)

16-bit architecture; instructions are always 16 bits long

2 instruction format categories: 3-bit, 6-bit

First bit in instruction signals format category (3-bit or 6-bit)

32 general-purpose registers

32-bit general-purpose communication bus (keyboard input, display output, etc.)

Big-endian system

#### Formats

M: [3-bits: instruction][5-bits: Rd][8-bits: I]

I: [3-bits: instruction][13-bits: I]

N: [6-bits: instruction][5-bits: Rn][5-bits: N]

D: [6-bits: instruction][5-bits: Rn][5-bits: Rm]

S: [6-bits: instruction][5-bits: Rn][2-bits: N][3-bits: F] See F instruction; allows additional execution conditions

F: [6-bits: instruction][5-bits: Rn][2-bits: Rm][3-bits: F] F-instructions check flags specified in F. Rm can only be X24-27 / AR-DR

#### Special registers

X29: PC - The program counter. Holds address of next instruction

X30 : LR - Link register. Used for branching

X28: SP - Stack pointer. Used for finding stack

X31: ZR - Zero register. Always holds zero and is non-writable

FL - Separate 5-bit register that stores flag outputs from the ALU and updates every falling clock edge

X24-X27: A, B, C, D - Limited Op Registers. Only registers that can used as second argument to F instructions. AR functions as accumulator as need be

#### 32-Bit Bus

Can be written to or read from

Functionality is implementation-defined; can be used to communicate with a hard drive, a keyboard, a small display, etc.

#### Flags

Zero (ZF)

Negative (NF)

Overflow (VF)

Carries (CF)

Always (AF)

#### Four 3-bit instructions

Meanings (ignoring first format bit):

0 means [varied]

1 means branch operation

0 00 PUTL: (M) I -> Rd | Loads an immediate into the lowest byte of a register (UPPER BYTE IS PRESERVED!!!)

0 01 PUTH: (M) I -< Rd | Loads an immediate into the higher byte of a register (LOWER BYTE IS PRESERVED!!!)

0 10 B: (I) PC += I<<2 | Adds signed int I to PC

0 11 BL: (I) LR = PC; PC += I<<2 | Adds signed int I to PC if zero flag is true

#### Thirty-two 6-bit instructions

First five bits of 6-bit instructions are ALWAYS destination register

Secondary register of ALL ALU ops must be X24-X27 (A-D)

Meanings (ignoring first format bit):

0 (leading) means ALU op

00 means bitwise op

01 means adder/longer math op (and last bit means set flags except for DIV)

100 means branch -> fourth means with link, fifth means relative or absolute branch

101 means register values are copied

11 means memory operation

1 00000 ASH: (N) Arithmetic SHift (both ways). Rn << N | See special references

1 00001 LSH: (N) Logical SHift (both ways). Rn <<< N

1 00010 AND: (F) Rn & Rm

1 00011 NAND: (F) !(Rn & Rm)

1 00100 NOT: (S) (A, B) Flips bits of specified register; flips bits of specified register assuming 2’s complement //

1 00101 FST: (S) Sets flags based on specified register (same as SUBS Rn, 0)

1 00110 ORR: (F) Rn | Rm

1 00111 XOR: (F) Rn ^ Rm

1 01000 ADD: (F)

1 01001 ADDS: (F)

1 01010 SUB: (F)

1 01011 SUBS: (F)

1 01100 MUL: (F)

1 01101 MULS: (F)

1 01110 UDIV: (F)

1 01111 SDIV: (F)

1 10000 BR: (S) (A, B) PC = Rn; LR = PC, PC = Rn | Branch Register; Branch Register with Link //

1 10001 SWAP: (D) Mem[Rn] ⇔ Rm | Atomically swaps Rm and memory at address Rn (NOT YET IMPLEMENTED)

1 10010 STRB: (D) Mem[Rn](Lo) = Rm(Lo) | Places the low byte of Rm in memory

1 10011 LDRB: (D) Rm(Lo) = Mem[Rn](Lo) | Loads the byte at Rn into the low byte of Rm

1 10100 MVT: (F) Copies value from Rn to Rm conditionally

1 10101 MVF: (F) Copies value from Rm to Rn conditionally

1 10110 MOV: (D) Copies value from one register to another (explicit, not pseudo-op)

1 10111 OUT: (S) (A, B, C, D) Writes the full value of Rn to the lower 16 bits of the 32-bit bus; writes the full value of Rn to the upper 16 bits of the 32-bit bus; gets the lower 16 bits of the 32-bit bus; gets the upper 16 bits of the 32-bit bus.

1 11000 PUSH: (S) SP -= 2; Mem[SP]=Rn

1 11001 POP: (S) Rn = Mem[SP]; SP += 2

1 11010 STRC: (F) Mem[Rn] = Rm | STore Register Conditionally (Bottom byte only)

1 11011 LDRC: (F) Rm = Mem[Rn] | LoaD Register Conditionally (Bottom byte only)

1 11100 STR: (D) Mem[Rn] = Rm | STore Register; stores value in Rn at address in Rm

1 11101 LDR: (D) Rm = Mem[Rn] | LoaD Register; loads value at address in Rm to Rn

1 11110 STRO: (N) Mem[Rn+N] = AR | STore Register with Offset

1 11111 LDRO: (N) AR = Mem[Rn+N] | LoaD Register with Offset

#### Condition Codes

eq - EQual. Checks ZF: 000

ov - OVerflows. Checks VF: 001

lt - Less Than. Checks ZF and NF: 010

le - Less than or Equal to. Checks ZF and NF: 011

gt - Greater Than. Checks ZF and NF: 100

ge - Greater than or Equal to. Checks ZF and NF: 101

cs - CarrieS. Checks CF: 110

al - ALways. Checks AF: 111

Custom: User specifies a sequence of five bits corresponding to flags to check.

### Steep Assembler Documentation

Steep is the standard assembler for the N-TEA architecture.

All N-TEA Assembly is preprocessed, expanded, resolved, then translated directly.

Vertical bars means Assembly Instruction, curly braces means IR Instruction (explicit translation)

During expansion, uses of limited register aliases in non-conditional format are replaced with regular register name.

During expansion, conditional format instructions called without a condition are explicitly given al conditional argument.

**IMPORTANT: All Steep assembly programs MUST have an entry point label “\_start:”**

#### Data Transfer Instructions

PUT - Will put an immediate into a register. If immediate is greater than 255, the higher byte of the value will be loaded, then the register will be left-shifted 8 bits, then the lower byte will be loaded. If a label is given instead of a value, the absolute address of the label will be loaded.

Example: |PUT X0, 23| becomes {PUTH X0 0; PUTL X0 23}, |PUT X1, 256| becomes {PUTH X0 1; PUTL X0 0}

MOV - Will transfer the value of one register to another register. If a condition is given, will use MVT and MVF to conditionally move from limited register to regular register or vice-versa. If an immediate is given, becomes PUT.

Example: |MOV X17, A| becomes {MOV X17 X24}, |MOV.eq A, X4| becomes {MVT X4 A eq}, |MOV X2, C| becomes {MVF X2 C al}, |MOV X1, 5| becomes {PUTH X1 0; PUTL X1 5}

MVT.cc - Explicit form of MVT (see MOV)

MVF.cc - Explicit form of MVF (see MOV)

INL - Conditionally records 16-bit bus value to register.

Example: |IN.eq X2| becomes {OUT X2 C eq}

INH - Conditionally records 16-bit bus value to register.

Example: |IN.eq X2| becomes {OUT X2 D eq}

OUTL - Conditionally writes register value to 16-bit bus.

Example: |OUTL X5| becomes {OUTL X5 A al}

OUTH - Conditionally writes register value to 16-bit bus.

Example: |OUTH X5| becomes {OUTH X5 B al}

#### Memory Transfer Instructions

STR - Stores register value in memory. Can accept condition if data to store is in limited register. Can use offset if no condition or second register given.

Example: |STR X3, X2| becomes {STR X3 X2}, |STR.cs X0, D| becomes {STRC X0 D cs}, |STR SP, 18| becomes {STRO X27 18}

STRC.cc - Explicit form of STRC.

STRO - Explicit form of STRO.

STRB - Explicit form of STRB.

LDR - Loads memory value into register. Can accept condition if destination register is limited. Can use offset if no condition or second register given.

Example: |LDR X3, X4| becomes {LDR X3 X4}, |LDR.le C, B| becomes {LDRC X26 B le}, |LDR X2, 8| becomes {LDRO X2 8}

LDRC.cc - Explicit form of LDRC.

LDRO - Explicit form of LDRO.

LDRB - Explicit form of LDRB.

SWAP - Explicit form of swap.

PUSH - Decrements stack pointer and puts value of register into stack.

POP - Puts top value of stack into register and increments stack pointer.

#### Branch Instructions

B - Branches to specified address. If immediate given, adds that immediate to the program counter. If register given, sets the program counter to that register. Can accept condition if register given instead of immediate.

Example: |B 32| becomes {B 32}, |B X16| becomes {BA X16 A al}, |B.ne B| becomes {BA X25 A ne}

BL - Sets link register and branches to specified address. Functions exactly the same as B but uses BL variations instead.

BR - Explicit form of BR.

BLR - Explicit form of BLR.

CALL - Pseudonym for BL.

RET - Pseudonym for BR.al LR.

SVC - Pseudonym for BL \_svc

#### Arithmetic Instructions

LSL - Performs an in-place logical shift left on value in specified register.

Example: |LSL X2, 4| becomes {LSH X2 4}

LSR Performs an in-place logical shift right on value in specified register.

Example: |LSR B, 3| becomes {LSH X25 -3}

ASR - Performs an in-place arithmetic shift right on value in specified register.

Example: |ASR X13, 11| becomes {ASH X13 -11}

NOT - Flips bits of specified register.

Example: |NOT X12| becomes {NOT X12 A al}

NEG - Flips sign of specified register assuming 2’s complement.

Example: |NEG B| becomes {NOT X25 B al}

All other operations - No special adjustments for expansion.

### Kettle OS

#### 

#### Functions of Kettle OS

Kettle OS consists of code attached to the beginning of a program before assembly takes place. It is loaded into the beginning of memory, and provides several useful subroutines for programs to use.

When a branch is made to \_svc subroutine, arguments are checked and run. X7 is used to specify which subroutine to use.

* 0: Branches to \_start.
* 1: Calls BRK. Allocates amount of heap memory specified in X0 and returns address of first byte in X0.
* 2: Calls done. Ends the program during automatic execution of instructions.

Some routine names are restricted because of pre-existing definition. Includes:

* \_end
* \_brk
* \_done
* \_svc

Example code to add two 32-bit numbers

word\_add:

mov a, 2

mov x2, x0

mov x3, x1

add x2, a //First num lo addr

add x3, a //Second num lo addr

ldr x2, a //First num lo -> a

ldr x3, b //Second num lo -> b

adds a, b //Sum lo -> a

mov x4, a //Sum lo -> x4

ldr x0, a //First num hi -> a

ldr x1, b //Second num hi -> b

add a, b

mov b, 1

add.ov a, b //If lo carried out, add one to hi

str x0, a //Hi sum -> first num hi

str x2, x4 //lo sum -> first num lo

ret //X0 now points to hi of sum

\_start:

mov a, 0b1000000000000000 //Lo

mov b, 0b0100000000000000 //Hi

push a

push b

mov x1, sp //Store address of second var in x1

push a

push b

mov x0, sp //Store address of first var in x0

call word\_add

mov a, 2

mov x1, x0 //Hi addr in x0

add x1, a //Lo addr in x1

ldr x0, x9 //Hi in x9

ldr x1, x10 //Lo in x10

mov x7, 2

svc

#### Kettle OS Source

//Set up branch table

MOV B, \_end

PUSH B

MOV B, \_start

PUSH B

MOV B, \_brk

PUSH B

MOV B, \_done

PUSH B

//Begin SVC loop

\_svc:

MOV A, 0xfffe //Bottom of stack

MOV C, 1

ADD x7, C

MOV C, 2

MUL x7, C //Offset from bottom of stack

MOV C, x7

SUB A, C //Address of subroutine to branch to

LDR A, B

BR B //Go to subroutine

\_brk:

MOV A, 0xfffe

LDR A, B

MOV D, B

MOV C, X0

ADD B, C

STR A, B

MOV X0, D

RET

\_done:

MOV x24, 0xffff

OUTL x24

OUTH x24

b \_done

### Planned Featuers

* Replace ACC instruction with something useful (PUTL and PUTH)
* Include STRB and LDRB instructions (DONE)
* Add functionality to assembler to enable static data storage
* Replace GF flag with some other flag (OV)
* Include atomic STR and LDR instructions
* Add support for static data section
* Remove relative register branch Instructions (DONE)
* Alter branch instruction to ignore last bit (instructions always aligned to 2 bytes) (bit-shift one byte) (DONE)
* Move system brk pointer to end instead of stack (DONE)
* Move branch table for OS to data section
* Allow exporting/importing of both assembly and compiled binary files
* Create proper linker for resolving references between compiled object files
* Create offline N-TEA emulator (do last so I don’t have to update two implementations of the same changing specifications)

### Known Bugs

* When using label with PUT instruction, upper half of register is not cleared. Also limits value of label branch to 255. Can be fixed in expansion or in resolution stage (EDIT: Alterations made to PUT instruction have since fixed this bug).

### Special References:

ASH: Arithmetic SHift. Shifts the given register arithmetically according to N. N is given as a signed integer -16 through 15, where if N >= 0, N += 1. The number is shifted left if N is positive and right if N is negative (check the first bit of N to determine right or left shift). 1 is added to positive or zero values of N to allow for full 16-bit left shift, assuming that it is unnecessary to be able to shift by 0 either direction.