ByteFrost Development Log - Addressing Modes Proposal Roadmap

July 13, 2025

Overview

In this development log, we shall outline the hardware and software implementation roadmap to implement the proposal. Also included are helpful reference materials such as the list of new control signals and hardware schematics of all new or modified hardware.

Instruction Updates

Instruction Operands Updates

There are `3` new instruction operands in this proposal. They are:

Operand	Size (in bits)	Bit Locations	Semantics
`(AR) L/H`	`1`	`5`	Whether the high or low byte of an AR operand is used (relevant for writing or reading a data word from / to the Data Bus)
`ARSrc`	`2`	**Special Case; See Below**	Specify which AR to read from (write to the Address Bus)
`ARDest`	`2`	`7:6`	Specify which AR to write to

Note: The `ARSrc` bit location depends on the opcode of the current instruction in the following way:

1. If the opcode is `0x1a` (of the `MAG` instruction), then `ARSrc` is in bits `9:8`.
2. Otherwise, `ARSrc` is in bits `{5, 0}` (where `0` is the lsb of the opcode).

- In both cases, the higher bit is the msb of `ARSrc` (`9` or `5`).
- This logic is implemented in hardware as **Decode - `ARSrc` Operand**.

New Instruction Operand Values

1. `(AR) L/H`

Bit 0 (location: `5`)	Meaning
`0`	Low Byte of an AR
`1`	High Byte of an AR

2. `ARSrc` and `ARDest`

Bit 1 (location: `ARSrc`: `9` or `5`; `ARDest`: `7`)	**Bit 0 (location: `ARSrc`: `8` or `0`; `ARDest`: `6`)**	**Meaning**
`0`	`0`	`PC` (**Note:** When `ARSrc` is `PC`, then `PC` is `{PC[H], PC[L]}`; when `ARDest` is `PC`, then `PC` is `{DHPC, PC[L]}`!)
`0`	`1`	`DP`
`1`	`0`	`SP`
`1`	`1`	`BP`

New / Updated Instruction List

There are `6` new instructions, which require a total of `9` opcodes, and `4` updated instructions.

Note: There is only `1` instruction that is currently being removed, which is `LSP` (opcode `0x14`), as it is eclipsed by the new `LDA` instruction.

Instruction	New / Modified	Semantics	Example Usage	Opcode Assignment
`PUSH`	Modified	`SP--; *SP = Rs1`	`PUSH R2`	`0x0e`
`POP`	Modified	`Rd = *SP; SP++`	`POP R2`	`0x0f`
`JSR`	Modified	`SP--; *SP = PC[H]; SP--; *SP = PC[L]; PC = DP`	`JSR`	`0x10`
`RTS`	Modified	`SP++; DHPC = *SP; SP--; PC[L] = *SP and PC[H] = DHPC; SP++; SP++;`	`RTS`	`0x11`
`LDWL`	New	`Rd = *(ARSrc + Imm)`	`LDWL R2, 16(%SP)`	`0x14`
`LDWH`	New	`Rd = *(ARSrc + Imm)`	`LDWH R3, -16(%DP)`	`0x15`
`SDWL`	New	`*(ARSrc + Imm) = Rs`	`SDWL R3, -4(%SP)`	`0x16`
`SDWH`	New	`*(ARSrc + Imm) = Rs`	`SDWH R1, 37(%BP)`	`0x17`
`MAAL`	New	`ARDest = ARSrc + Imm`	`MAAL %DP, %SP, #-1`	`0x18`
`MAAH`	New	`ARDest = ARSrc + Imm`	`MAAH %DP, %DP, #-1`	`0x19`
`MAG`	New	`Rd = ARSrc[L/H]`	`MAG R1, %SP[H]`	`0x1a`

Instruction	New / Modified	Semantics	Example Usage	Opcode Assignment
`LDA`	New	`ARDest[L/H] = Imm`	`LDA %DP[L], #0x54`	`0x1b`
`MGA`	New	`ARDest[L/H] = Rs1`	`MGA %BP[L], R2`	`0x1c`

Note: Instructions `LDWL` and `LDWH`, `SDWL` and `SDWH`, and `MAAL` and `MAAH` will be implemented as single ByteFrost Assembly instructions (i.e., `LDW`, `SDW`, and `MAA`) - the ByteFrost Assembler will decide based on the used `ARSrc` operand which ISA instructions to use.

Hardware Changes

Implementing this proposal requires implementing the following hardware changes:

1. **Decode - `ARSrc` Operand**

This is a combinational logic module to read the `ARSrc` operand from the current instruction string (stored in the instruction registers).

Inputs:

1. `Opcode_MAG` (active low signal from opcode decoder for opcode `0x1a`).
2. `INSTR[5]`
3. `INSTR[0]`
4. `INSTR[9:8]`

Outputs:

1. `ARSrc` (2 bits)

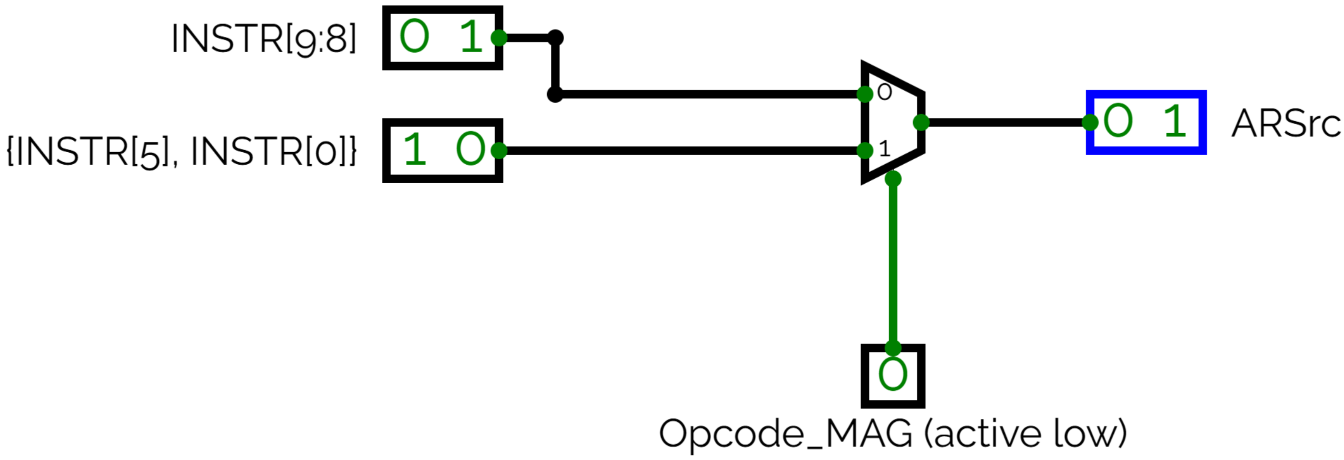
Logic:

1. If opcode == `MAG_OPCODE (0x1a)`:
    1. `ARSrc = INSTR[9:8]`.
2. Else:
    1. `ARSrc = {INSTR[5], INSTR[0]}`.

Truth Table:

`Opcode_MAG_0x1a` (active low)	Output (2 bits)
`0` (active)	`INSTR[9:8]`
`1` (inactive)	`{INSTR[5], INSTR[0]}`

Schematic:



2. PC Logic Revision

1. Remove the `PC Ld Hi` active low signal and the combinational logic that generates it.

- **Note:** This is replaced by the **AR Data Bus Load Enable**'s `PC[L] + PC[H] = DHPC Load Enable` active low signal.

2. Rename the `PC Ld Lo` active low signal to `PC Ld Branch` (this is the signal generated in the Branch (v2.0) in slide 26 of the CPU v2 PowerPoint).
3. Send the `PC Ld Branch` active low signal as an input to the **AR Data Bus Load Enable** LUT.
4. The `DHPC`'s load enable signal comes from the **AR Data Bus Load Enable** LUT, named `DHPC Load Enable`, after it has been NOR'd with the ByteFrost clock signal (since this signal goes to the `DHPC`'s clock signal input; i.e., it's a load "trigger").
5. The `PC[L]` and `PC[H]` load enable signals come from the **AR Data Bus Load Enable** LUt, named `PC[L] + PC[H] = DHPC Load Enable`.

- `PC[L] + PC[H] = DHPC Load Enable` is active low and goes to both `PC[L]` and `PC[H]`'s load enable inputs; as these counters have load enables, this signal does **NOT** have to be NOR'd with the ByteFrost clock signal.

6. The `PC Out` control signal is now an input of the **ARSelect** LUT. The `74HC245` "Read PC Low" tristate IC should be removed, as the `PC` may only write to the Address Bus (writing the `PC` to the Data Bus involves first writing the `PC` to the Address Bus).

- The new signal that controls whether the `PC` writes to the Address Bus is the active low `PC Output Enable` signal that is outputted by the **ARSelect** LUT.

3. **AR Data Bus Load Enable** LUT

Inputs:

EEPROM Address Bit	Input
7	`PC Ld Branch` (**active low**) signal
6	*loadAR* (control signal 17)
5	*PC Load* (control signal 8)

EEPROM Address Bit	Input
`4`	*loadARHorL* (control signal `18`)
`3`	`Opcode_MAA` (**active low**) signal
`2`	`AR (L/H)` instruction operand (`INSTR[5]`)
`1`	`ARDest[1]` instruction operand (`INSTR[7]`)
`0`	`ARDest[0]` instruction operand (`INSTR[6]`)

Note: The active low signal `Opcode_MAA` is generated by AND'ing the active low signals for opcodes `Opcode_MAAL_0x18` and `Opcode_MAAH_0x19`.

Outputs:

EEPROM Data Output Bit	Output
`7`	`BP[H] Load Enable` (**active low**)
`6`	`BP[L] Load Enable` (**active low**)
`5`	`SP[H] Load Enable` (**active low**)
`4`	`SP[L] Load Enable` (**active low**)
`3`	`DP[H] Load Enable` (**active low**)
`2`	`DP[L] Load Enable` (**active low**)
`1`	`DHPC Load Enable` (**active low**)
`0`	`PC[L] + PC[H] = DHPC Load Enable` (**active low**)

Note: At most 1 of the outputs will be `0`; all others will be `1`.

Note: The following signals need to be NOR'd with the ByteFrost clock signal before being used as load enables (i.e., load enable signals need to be NOR'd if the target register doesn't have a load enable input; in that case, these serve as load triggers).

1. `BP[H] Load Enable` (**active low**)
2. `BP[L] Load Enable` (**active low**)
3. `DP[H] Load Enable` (**active low**)
4. `DP[L] Load Enable` (**active low**)
5. `DHPC Load Enable` (**active low**)

4. **ARSelect** LUT

Inputs:

EEPROM Address Bit	Input
`7`	`Bus Grant`

EEPROM Address Bit	Input
`6`	`FetchCycle`
`5`	*PC Out* (control signal `9`)
`4`	*SP Out* (control signal `14`)
`3`	*TmpARWrite* (control signal `20`)
`2`	`Opcode_MAG_LDW_SDW_MAA`
`1`	`ARSrc[1]` instruction operand
`0`	`ARSrc[0]` instruction operand

Outputs:

EEPROM Data Output Bit	Output
`7`	`1` (**unused**)
`6`	`1` (**unused**)
`5`	`1` (**unused**)
`4`	`TmpAR Output Enable` (**active low**)
`3`	`BP Output Enable` (**active low**)
`2`	`SP Output Enable` (**active low**)
`1`	`DP Output Enable` (**active low**)
`0`	`PC Output Enable` (**active low**)

5. **AddressByteSelect**

Inputs:

1. *AddressHorL* (control signal `22`)
2. *AddressBusToDataBus* (control signal `21`)
3. `(AR) L/H` instruction operand (`INSTR[5]`)
4. `Opcode_MAA_JSR` (**active low**)

- **Note:** This signal is generated by AND'ing the active low outputs of the opcode decoder for opcodes of `MAA` (`0x18`, `0x19`) and `JSR` (`0x10`).

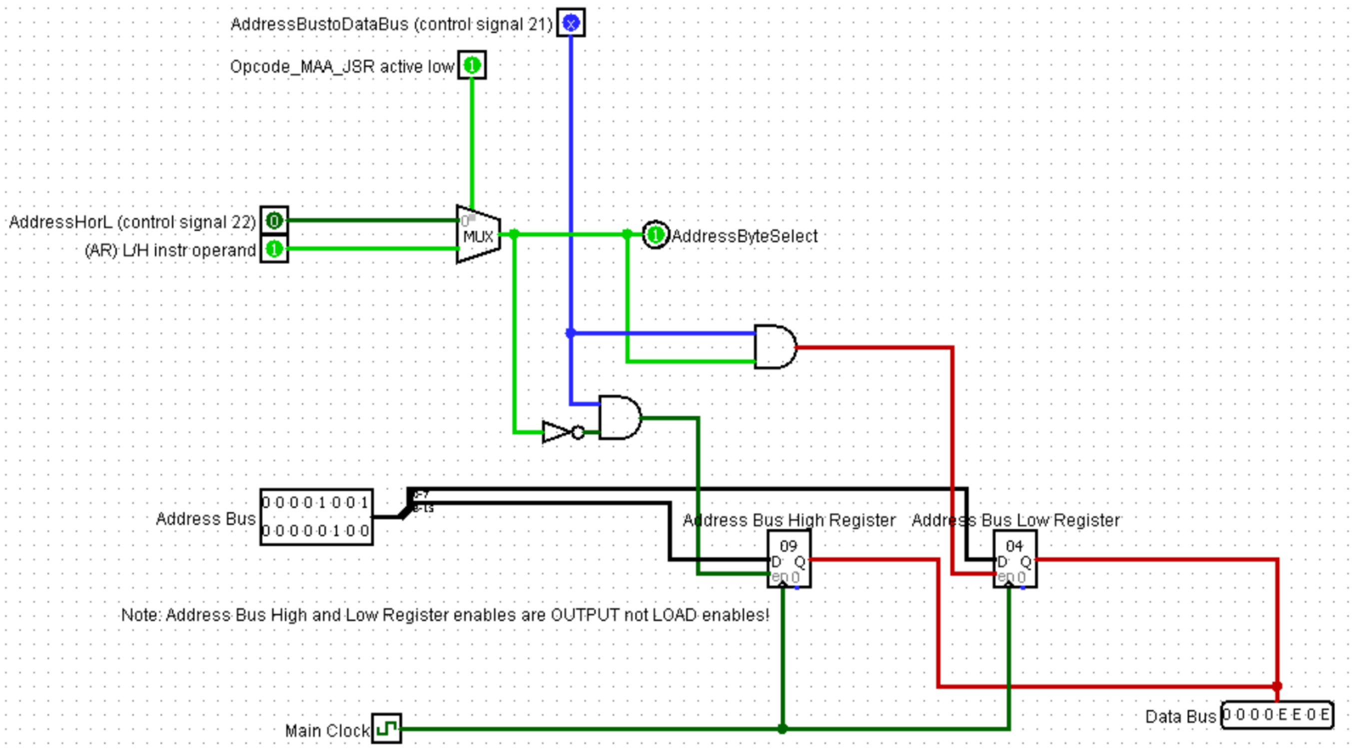
Outputs:

Logic:

1. If *AddressBusToDataBus* is active (high):
    1. If Opcode == `MAA_Opcode_0x18_x19` OR Opcode == `JSR_Opcode_0x10`:
        1. If *AddressHorL* is active (high):
            1. Return `Address Bus High Register Output Enable`.

            2. Else:
                1. Return `Address Bus Low Register Output Enable`.
        2. Else:
            1. Return `Address Bus {(AR) L/H instruction operand} Register Output Enable`.
    2. Else:
        1. Return `None`.

Schematic:



Control Signals

Bit	Control Signal
`23`	Not Used.
`22`	*AddressHorL*
`21`	*AddressBusToDataBus*
`20`	*TmpARWrite*
`19`	*TmpARRead*
`18`	*loadARHorL*
`17`	*loadAR* (formerly: Load Special Pointer)
`16`	*Stack Pointer Increment / Decrement* (`0`: decrement / `1`: increment)
`15`	*Stack Pointer Count*
`14`	*SP Out* (formerly: RAM Address Select)
`13`	*Use Rd as Source*

Bit	Control Signal
`12`	*Lower Address Register Load*
`11`	*Mem Write*
`10`	*Mem Read*
`9`	*PC Out*
`8`	*PC Load*
`7`	*PC Advance*
`6`	*Program Register H Write to Bus*
`5`	*Register File Input Enable*
`4`	*Register File Output Enable*
`3`	*Register File Output Select* (`0`: Rs1 / `1` Rs2)
`2`	*ALU output enable*
`1`	*ALU load register A*
`0`	*ALU load register B*

Implementation Roadmap

To implement the proposal, the following things must be done:

1. Hardware
    1. **`ARSrc` Operand Decode Logic**
    2. PC Logic Revision
    3. **AR Data Bus Load Enable LUT**
    4. **ARSelect** LUT
    5. **AddressByteSelect** combinational logic and registers.
2. Microcode
    1. Update microcode for `PUSH`, `POP`, `JSR`, and `RTS`
    2. Overwrite microcode for `LSP` and add microcode for `LDWL`, `LDWH`, `SDWL`, `SDWH`, `MAAL`, `MAAH`, `MAG`, `LDA`, and `MGA`.
3. Software
    1. ByteFrost Assembler v2:
        1. Ensure all instructions in the ISA have a corresponding representation in the assembler code.
            1. Remove `LSP`.
            2. Update `JSR`'s ISA instruction to take no operands.
            3. Add `LDWL`, `LDWH`, `SDWL`, `SDWH`, `MAAL`, `MAAH`, `MAG`, `LDA`, and `MGA` ISA instructions.
        2. Update the set of ByteFrost Assembly instructions:
            1. Remove `LSP`.
            2. Add `LDW`, `SDW`, `MAA`, `MAG`, `LDA`, and `MGA`.
            3. Update `CALL` to work using the new `JSR` ISA instruction.

3. Update the parser to recognize new syntax
    1. AR token (i.e., `%AR` -> `%PC`, `%DP`, `%SP`, and `%BP`)
    2. AR Offset token (`Imm(AR)`) used in `LDW` and `SDW`, e.g., `-5(%DP)` or `28(SP)`. --> Note that in this case the immediate would not have a `#` in front, so perhaps the regular expression would be `Number(AR)`
2. LUT Generator.
    1. Write a program that generates the look up tables for the EEPROMs for **AR Data Bus Load Enable** and **ARSelect**.

Dependency Graph

ARSelect LUT Implementation Checklist

Implementing the **ARSelect** LUT requires the following:

1. Address Bus Arbiter
    1. Remove the Address Bus Arbiter.
    2. Replace the Address Bus Arbiter with the **ARSelect** LUT EEPROM, and ensure that all signals that were outputted by the Address Bus Arbiter are replaced with those outputted by **ARSelect**:
        1. Replace `Data Pointer OE` (output of the Address Bus Arbiter) with the **ARSelect** LUT's `DP Output Enable` signal.
        2. Replace `Stack Pointer OE` (output of the Address Bus Arbiter) with the **ARSelect** LUT's `SP Output Enable`.
        3. Replace `Program Counter OE` (output of the Address Bus Arbiter) with the **ARSelect** LUT's `PC Output Enable`.
2. New Registers
    1. Replace the `74HC574 Stack Ptr Page` register with two `74HC169` U/D counters.
    - **Note:** These counters have load enables. This means that instead of using the current **Load Special Pointer** `Ld Stack Ptr Hi` signal that serves as a load trigger, we can use the load enable signal generated (eventually) by the **AR Data Bus Load Enable** LUT. However, until that is implemented, it is fine to still use the `Ld Stack Ptr Hi` load trigger signal, but eventually it should be removed.
    2. Add the Base Pointer (`BP`) AR.
        1. Connect the `BP Output Enable` output of **ARSelect** to this AR's output enable.
    3. Add the TmpAR register.
        1. Connect the `TmpAR Output Enable` output of **ARSelect** to this AR's output enable.
3. Input requirements
    1. Create the active high `Opcode_MAG_LDW_SDW_MAA` signal.
    2. Add the *TmpARWrite* control signal.
    3. Fully implement the **`ARSrc` Operand Decode Logic**.
4. Output requirements
    1. Ensure all used outputs of the **ARSelect** LUT are properly connected as output enables for ARs `PC`, `DP`, `SP`, `BP`, and `TmpAR`.
5. PC Out Logic Revision
    1. Remove the `Read PC Low 74HC245` chip that allows writing `PC[L]` to the Data Bus.
    - The *PC Out* control signal becomes an input of the **ARSelect** LUT; it should **NOT** be used as an output enable directly.

  - **Note:** Doing this would break the `JSR` and Branch Relative instructions as they are currently implemented.

Once **ARSelect** is fully implemented, the following instructions will cease to work:

1. `JSR`.

- `JSR` will break because the `Read PC Low 74HC245` chip that allows writing `PC[L]` to the Data Bus will be removed (repurposing of the *PC Out* control signal).

2. Branch Relative.

- This is less important - Branch Relative is deprecated anyway, and the ByteFrost Assembler doesn't use it (so no binary files generated by the Assembler v2 will break).