CS617 | Chapter 01 | NOTES

Unsigned 32-bit Numeric Library and Calculator

Alex Barragan and Sol | Xandrian Labs

Approved harmonized edition - 3 October 2026

Contents

CS617 Chapter 01 **1**
Unsigned 32-bit Numeric Library and Native Calculator . . . 1
Purpose . . . 1
1. Numerical model . . . 1
2. Reuse and provenance . . . 2
3. Independent validation before integration . . . 2
4. Decimal input boundary . . . 2
5. Decimal output boundary . . . 3
6. Calculator application . . . 3
7. Calculator arithmetic behavior . . . 3
8. Library policy versus application policy . . . 4
9. Evidence and deliverables . . . 4
10. Acceptance criteria . . . 4
11. Explicit non-goals . . . 5
12. Chapter boundary . . . 5

CS617 Chapter 01

Unsigned 32-bit Numeric Library and Native Calculator

Edition: CS617 approved harmonized teaching reconstruction (2026-10-03). It is derived from accepted Chapter 01 Experience, not a verbatim original assignment. No new historical grading obligations are asserted.

Purpose

Build a reusable unsigned 32-bit numerical layer for the HD61700 and demonstrate that layer in a small native ROM-resident calculator for the Casio PB-2000C. The PB-2000C is the course development and validation target; this assignment does not assert compatibility with other machines.

The chapter has two inseparable products:

```
1. a reusable unsigned 32-bit numeric library
2. an interactive calculator that consumes that library
```

The application is not the place where arithmetic is first proven.

The chapter principle is:

> One implementation, many proofs and many consumers.

This document defines the **WHAT**. Register allocation, exact ROM placement, choice of algorithms, relocation strategy, screen coordinates, and the sequence of intermediate experiments belong to the **HOW**.

---

1. Numerical model

The chapter uses unsigned 32-bit values:

```
0 <= x <= 4294967295
```

Required arithmetic semantics are:

```
addition       modulo 2^32
subtraction    modulo 2^32
multiplication low 32 bits, modulo 2^32
integer divide unsigned quotient
remainder      unsigned remainder
```

Division and remainder require a nonzero divisor at the reusable arithmetic boundary. The calculator must prevent a zero divisor from reaching an unchecked primitive and must report the condition cleanly to the user.

The reusable layer shall expose capabilities equivalent to:

```
UADD32        u32 + u32
USUB32        u32 - u32
UMUL32        u32 * u32 -> low u32
UDIV32NZ      u32 / u32 -> quotient and remainder, divisor nonzero
UDIV3216NZ    u32 / u16 -> quotient and remainder, divisor nonzero
UMOD32NZ      u32 % u32, divisor nonzero
UMOD3216NZ    u32 % u16, divisor nonzero
```

These are project-facing capability names. The assignment does not prescribe their register ABI or internal implementation.

---

2. Reuse and provenance

A reusable primitive has one authoritative implementation source.

Standalone tests and applications shall consume that implementation rather than carry private arithmetic copies.

Historical Casio code may be recovered and adapted where suitable. When a student claims an exact or near-exact historical transplant, provenance must remain explicit and the claim must be supported by appropriate evidence.

New adapters or compositions are permitted and should be identified as such. Do not describe newly written wrapper/composition code as historical donor code.

A student is not required to reproduce any particular archaeological sequence. What matters is that the final reusable contracts are understood, independently validated, and honestly documented.

---

3. Independent validation before integration

Do not write the final calculator first and then infer that every primitive must be correct because the calculator appears to work.

Each promoted capability needs an independent validation checkpoint sufficient for the domain in which later code will use it.

A checkpoint should preserve enough evidence to connect:

```
source
    -> assembler interpretation/listing
    -> exact executable ROM
    -> predicted behavior
    -> observed behavior
    -> scoped conclusion
```

Standalone self-test ROMs are a natural solution, but the assignment does not require students to reproduce Sol's exact number or ordering of test ROMs.

For historical transplants, byte identity is useful evidence when the student's claim depends on byte-identical relocation.

Expected arithmetic results should come from an independent oracle, not from the routine under test generating its own expected answers.

---

4. Decimal input boundary

The calculator receives decimal digits incrementally, so the reusable layer shall provide a capability equivalent to:

```
UDECAPP(current, digit)
```

where `digit` is numeric `0..9`.

On success:

```
new_value = current * 10 + digit
status    = success
```

If the result would exceed `4294967295`:

```
current value is preserved
status reports overflow
```

The implementation must reject overflow before ordinary modulo-2^32 arithmetic silently wraps the value.

A full buffered decimal parser is not required.

---

5. Decimal output boundary

The reusable layer shall provide a capability equivalent to:

```
UFMT32DEC(value)
```

which produces ordinary decimal ASCII with no leading zeroes.

Required visible behavior includes:

```
0          -> "0"
123        -> "123"
4294967295 -> "4294967295"
```

The formatted representation must be terminated or length-described well enough for a caller to consume safely. The application must not contain a private second implementation of decimal formatting.

The exact formatting algorithm is not prescribed.

---

6. Calculator application

Build a standalone ROM application around three logical fields:

```
[ op1 ][op][ op2 ]
```

with:

```
op1, op2 : unsigned u32
op       : +  -  *  /  %
```

Required interaction:

- decimal digits append to the active operand;
- selecting an operator completes/activates the appropriate fields;
- selecting another operator before `op2` is complete may replace the operator;
- the completed expression can be evaluated from the keyboard;
- after evaluation, the result becomes the new `op1`, allowing chained work;
- a backspace/edit action removes digits and can retreat across empty fields;
- a digit entered after a completed result starts a fresh expression;

- the active field/append position is visibly identifiable;
- `BRK` retains its normal device-level restart/reinitialization role.

The final frozen implementation accepted both `=` and `EXE` as evaluation keys, but exact key mapping and screen geometry are implementation choices unless the instructor specifies them separately.

The `%` operation must be accessible. Implementing the complete historical shift-key subsystem is not required merely to satisfy this calculator.

---

7. Calculator arithmetic behavior

The application exposes the library's unsigned semantics rather than inventing signed behavior.

Representative consequences include:

```
0 - 1                -> 4294967295
4294967295 + 1       -> 0
4294967295 * 2       -> 4294967294
7 / 2                -> 3
7 % 3                -> 1
```

A zero divisor must produce a user-visible error instead of entering an unchecked divide/remainder primitive.

Decimal input above `4294967295` must be rejected without corrupting the current operand.

---

8. Library policy versus application policy

The reusable numerical layer should implement general numerical contracts. Application-specific user-interface policy belongs in the calculator unless it is genuinely part of a reusable numeric contract.

For example, decimal numeric overflow is a library concern. A decision about how many visible operand digits the calculator accepts before calling the library can be an application concern.

Do not distort an accepted general-purpose primitive merely to make one screen behavior easier.

---

9. Evidence and deliverables

The final submission should preserve, at minimum:

```
reusable library source
calculator source
assembler listing(s)
exact 64 KiB ROM image(s)
independent test predictions/oracles
runtime observations
provenance notes for recovered historical code
hashes or equivalent executable identities
concise final writeup
```

A listing and binary are experimental evidence, not disposable build products.

If a candidate fails, preserve that checkpoint before repairing it. The repaired version becomes a new checkpoint rather than rewriting the history of the old one.

---

10. Acceptance criteria

Chapter 01 is successful when:

1. unsigned 32-bit ADD, SUB, MUL, DIV and remainder capabilities exist with clearly stated contracts;
2. the library provides the narrower divide/remainder support needed by decimal conversion or an equivalent reusable design;

3. decimal digit accumulation rejects values above `UINT32_MAX` while preserving the old value;
4. decimal formatting handles the required u32 range, including zero and `4294967295`;
5. promoted capabilities have independent validation appropriate to their use;
6. historical provenance remains explicit wherever historical code is reused;
7. exact-transplant claims are supported by byte-identity evidence where applicable;
8. the calculator consumes the shared library rather than private arithmetic copies;
9. calculator `+`, `-`, `*`, `/`, and `%` behavior agrees with unsigned u32 semantics;
10. divide/remainder by zero is guarded at the application boundary;
11. decimal overflow is rejected without corrupting the operand;
12. chaining, editing/retreat, operator replacement, and fresh entry after a result behave consistently;
13. a complete 64 KiB ROM assembles cleanly and is identified together with its listing and source;
14. the final human-visible behavior agrees with the student's documented specification and test evidence.

---

11. Explicit non-goals

The reconstructed core assignment does not require:

```
signed arithmetic
floating-point arithmetic
arbitrary-precision integers
scientific notation
parentheses or expression parsing
operator precedence
memory registers
complete reproduction of the PB-2000C shifted-key subsystem
rewriting every numerical routine when a justified reusable historical surface exists
```

---

12. Chapter boundary

The student must reason from the required behavior to an implementation.

The assignment intentionally leaves open:

```
algorithm choice
register allocation
workspace layout
ROM placement
historical routine selection
wrapper design
test-corpus organization
number of intermediate ROMs
screen geometry
cursor implementation
keyboard adapter details
```

A strong solution explains why its choices satisfy the contracts and why its evidence supports its claims.