Claim: \( \sum_{i=a}^{b} A[i] \) is a loop invariant.

Pre':

\[
\text{while } \ i \neq b+1 \text{ do} \quad \sum := \sum + A[i]; \quad i := i + 1;
\]

Post': \( \sum = \sum_{i=a}^{b} A[i] \)
Claim: $\sum_{i=1}^{b+1} A[i] \geq 0$ is invariant

(* Pre: $a \leq b+1 \implies:\)

Proof of Claim

By induction on the number of iterations, $k$

$\implies:\)

Let $i := a$; sum := 0;

$i := i + 1$;

while $i \neq b+1$ do

sum := sum + A[i];

end;

Basis: $k = 1$, i.e. I holds at the beginning of the first iteration.

We already checked this

Ind. step: Assume I holds at the beginning of the $k$th iteration.

If the $k$th iteration is the last one, then done.

Else, $i \neq b+1$, so we need to show that $\{ I \land i \neq b+1 \} \implies I$.

Let $i$ and $\text{sum}$ be the values of the variables $i$ and
sum before the k-th execution of the loop, and let $i'$ and $\text{sum'}$ be the values of the variables $i$ and $\text{sum}$ after the k-th execution of the loop. Then, $i' = i + 1$ and $\text{sum'} = \text{sum} + A[i]$. So, we need to show:

\[
\begin{align*}
\text{sum} = \sum_{i=a}^{i=b+1} A[i] & \Rightarrow \text{sum'} = \sum_{i=a}^{i=b+1} A[i] \\
\text{sum} + A[i] & = \sum_{i=a}^{i=b+1} A[i] \\
\frac{1}{2} \sum_{i=a}^{i=b+1} A[i] & = \sum_{i=a}^{i=b+1} A[i]
\end{align*}
\]
Example (MIPS according to Patterson & Hennessy)
symbolic representation: add $1, $2, $5

decimal representation: \( 0 \) \( 17 \) \( 18 \) \( 18 \) \( 0 \) \( 32 \)

binary representation: 
\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

instruction format: 
\[
\begin{array}{cccccc}
op & rs & rt & rd & shamt & funct
\end{array}
\]
op: opcode (basic operation) 
rs: first register source operand
rd: register destination operand
rd: register destination operand
shamt: shift amount
funt: function code (chooses a variant of the operation in op)

• Stored program concept: single memory (store)
  for programs and data ("von Neumann" machines)
• Our target machine, TAM, is a two-store machine
  (a "von Neumann" machine)
Different architectures styles [Patterson & Hennessy 1998, p. 201]

- accumulator or "single-register", e.g., IBM 701
- memory - memory, e.g., DEC VAX and IBM 360, which is however partly load-store
- load-store: all operations occur in registers
- stack (!!!): [RAM is a stack machine]

Advantage: no register allocation problem!